SINTEF SimViewer

A simulation playback tool

PROJECT REPORT – GROUP 7

TDT4290 CUSTOMER DRIVEN PROJECT

Erling Børresen	*erlingbo@stud.ntnu.no*
Pavel Fatrdla	*pfatrdla@seznam.cz*
Erik Hidle	*erikhi@stud.ntnu.no*
Jiří Kadlec	*jir.kadlec@post.cz*
Rolf Klungsøyr	*klungsoy@stud.ntnu.no*
Jørgen Nystad	*jorgenys@stud.ntnu.no*

NORWEGIAN UNIVERSITY OF SCIENCE AND TECHNOLOGY

Preface

This report and associated software are the results of a project that started August 26th, 2008, lasting throughout the semester.

Our assignment, given by SINTEF Fisheries and Aquaculture, was to create a tool to visually playback previously computed simulations, using an existing library from their part, and add extensions allowing statistical analysis of the data if possible.

The simulations are of marine facilities and machinery, and are currently in use in the marine industry.

Trondheim, November 20th, 2008

Erling Børresen	Jiří Kadlec
Pavel Fatrdla	Rolf Klungsøyr
Erik Hidle	Jørgen Nystad

Contents

Preface i
Contents iii
Tables xi
Figures xii
Summary 1
Background 1
Objectives 1
Results 2
Chapter 1 Project Directive 3
1.1. Chapter introduction 4
1.2. Project mandate 4
1.2.1. Project name 4
1.2.2. Customer 4
1.2.3. Project background 5
1.2.4. Involved parties 5
1.2.5. Result goals 5
1.2.6. Project scope 5
1.2.7. External conditions 6
1.2.8. Economy 7
1.2.9. Duration 7
Chapter 2 Project Management 9
2.1. Chapter introduction 10

2.2. About Scrum 10
2.2.1. Why Scrum 12
2.3. Project plan 13
2.3.1. Structure of the project plan 13
2.3.2. Workload 16
2.3.3. Milestones and deadlines 17
2.3.4. Gantt diagrams 18
2.3.5. Progress report from XPlanner 18
2.4. Project team 18
2.4.1. Project manager 19
2.4.2. Test manager 19
2.4.3. Document manager 19
2.4.4. Customer contact 19
2.4.5. Supervisor contact 20
2.4.6. Meeting minutes manager 20
2.5. Risk analysis 21
2.6. Quality assurance 26
2.6.1. Communication 26
2.6.2. Documents 27
Chapter 3 Preliminary Studies 29
3.1. Chapter introduction 30
3.2. Gantt diagram 31
3.3. Resource management 31
3.3.1. Version control system 31
3.3.2. Document sharing 32
3.3.3. XPlanner 32
3.4. Connecting fhVis with SimViewer 34

3.4.1. JNI – Creating a connection 34
3.4.2. OGRE 34
3.5. Graphical user interface 36
3.5.1. AWT (Abstract Window Toolkit) 36
3.5.2. SWT (Standard Widget Toolkit) 36
3.5.3. Swing 36
3.5.4. AWT vs. Swing 37
3.5.5. SWT vs. Swing 37
3.5.6. Bridge between SWT and AWT/Swing 38
3.6. Charting 39
3.6.1. Requirements 39
3.6.2. Candidates 39
3.6.3. JFreeChart 40
Chapter 4 Requirements Specification 45
4.1. Chapter introduction 46
4.2. Overview 47
4.2.1. Proposed interaction with fhVis 47
4.2.2. User characteristics 48
4.2.3. Dependencies 48
4.3. Use cases 48
4.3.1. Textual use cases 49
4.3.2. Use case diagram 55
4.4. Requirements 56
4.4.1. Functional requirements 56
4.4.2. Non-functional requirements 62
4.5. Project backlog 65
Chapter 5 Sprint 1 67

5.1. Chapter introduction 68
5.2. Sprint backlog 68
5.3. Sprint Gantt chart 69
5.4. Design 69
5.4.1. Package and class diagrams 70
5.5. Development 77
5.5.1. Development of NativeBridge 77
5.5.2. Development of SceneWindow 79
5.5.3. Simulation controller framework 82
5.5.4. Simulation object list for GUI 84
5.6. Testing 85
5.7. Conclusion 88
Chapter 6 Sprint 2 89
6.1. Chapter introduction 90
6.2. Sprint backlog 90
6.3. Sprint Gantt chart 91
6.4. Design 91
6.4.1. Class diagrams 91
6.5. Development 98
6.5.1. Making SimViewer Vista compatible 98
6.5.2. Passing state data to fhVis 99
6.5.3. Development of simulation object tree 100
6.5.4. Simulation controller framework – Parser changes 104
6.6. Testing 108
6.7. Conclusion 114
Chapter 7 Sprint 3 117
7.1. Chapter introduction 118

7.2. Sprint backlog 118
7.3. Sprint Gantt chart 119
7.4. Design 119
7.4.1. Package and class diagrams 119
7.5. Development 128
7.5.1. Parser 128
7.5.2. Adding functionality to the simulation object tree 128
7.5.3. Charting 130
7.5.4. Following camera 131
7.5.5. Screen shot functionality 133
7.5.6. Log functionality 134
7.6. Testing 135
7.7. Conclusion 138
Chapter 8 Sprint 4 139
8.1. Chapter introduction 140
8.2. Sprint backlog 141
8.3. Sprint Gantt diagram 141
8.4. Design 142
8.4.1. Class diagrams 142
8.5. Development 149
8.5.1. Changes in charting framework 149
8.5.2. SimulationControl ↔ PlaybackControl 152
8.5.3. GUI changes 153
8.6. Testing 154
8.7. Conclusion 156
Chapter 9 After Work 157
9.1. Chapter introduction 158

9.2. Project report 158
9.3. Presentation 158
Chapter 10 User Guide 159
10.1. Chapter introduction 160
10.2. System requirements 160
10.2.1. Operating System 160
10.2.2. Software prerequisites 160
10.2.3. Hardware requirements 161
10.3. Startup dialog 162
10.3.1. Configuration file 163
10.3.2. Model output file 163
10.3.3. Path to resources 163
10.3.4. Time step 163
10.3.5. Start visualization 163
10.4. The simulation main window 164
10.4.1. Camera control 165
10.4.2. Keyboard focus 165
10.4.3. Capture screen 165
10.4.4. Simulation object list 165
10.4.5. Simulation speed control 166
10.4.6. Playback control 166
10.4.7. Camera options 166
10.4.8. Charting 167
Chapter 11 Evaluation 169
11.1. Chapter introduction 170
11.2. Internal process 170
11.2.1. What we did well 170

11.2.2. What we did not do well 171
11.2.3. What we learned 172
11.3. Customer 173
11.4. Supervisors 174
11.5. Further work 175
11.6. Suggestions for improvements 176
11.6.1. Lecture schedule 177
11.6.2. Information flow 177
11.6.3. Supervisors 178
11.6.4. Scrum vs. waterfall 178
11.6.5. Assigning students 179
Glossary 181
References 183
Appendix A Involved parties A-1
A.1. Project team A-2
A.2. Customer A-2
A.3. Supervisors A-2
Appendix B Project Plan Resources B-1
B.1. Time statistics B-2
B.1.1. Preliminary study B-2
B.1.2. Sprint 1 ..... B-3
B.1.3. Sprint 2 ..... B-4
B.1.4. Sprint 3 ..... B-5
B.1.5. Sprint 4 ..... B-7
B.1.6. After work B-8
Appendix C Extended Documentation C-1
C.1. XPlanner C-2

C.1.1. Main parts of XPlanner project (terminology) C-2
C.1.2. Structure of project and user interface.....C-3
C.1.3. Export functionality.....C-6
C.1.4. Technical information.....C-7

Tables

Table 2.1: Time spent in iterations 16
Table 2.2: Deadlines 17
Table 2.3: Risk analysis 25
Table 4.1: Project backlog 66
Table 5.1: Sprint 1 backlog 69
Table 5.2: Sprint 1 class descriptions 77
Table 6.1: Sprint 2 backlog 91
Table 6.2: Sprint 2 class descriptions 97
Table 7.1: Sprint 3 backlog 119
Table 7.2: Sprint 3 class descriptions 128
Table 8.1: Sprint 4 backlog 141
Table 8.2: Sprint 4 class descriptions 148

Figures

Figure 2.1: Scrum process 11
Figure 2.2: Overall Gantt diagram 18
Figure 3.1: Bar chart example output 41
Figure 3.2: XY chart example output 42
Figure 3.3: Pie chart example output 43
Figure 4.1: Use case diagram 55
Figure 5.1: Sprint 1 Gantt chart 69
Figure 5.2: Sprint 1 package diagram 70
Figure 5.3: Sprint 1 class diagram for simviewer.jni 70
Figure 5.4: Sprint 1 class diagram for simviewer.data 71
Figure 5.5: Sprint 1 class diagram for simviewer.visualize 71
Figure 5.6: Sprint 1 class diagram for simviewer.gui part 1 72
Figure 5.7: Sprint 1 class diagram for simviewer.gui part 2 73
Figure 5.8: Simulation controller buffering 83
Figure 6.1: Sprint 2 Gantt chart 91
Figure 6.2: Sprint 2 class diagram for simviewer.jni 92
Figure 6.3: Sprint 2 class diagram for simviewer.data 93
Figure 6.4: Sprint 2 class diagram for simviewer.visualize 94
Figure 6.5: Sprint 2 class diagram for simviewer.gui part 1 95
Figure 6.6: Sprint 2 class diagram for simviewer.gui part 2 96
Figure 6.7: Activity diagram for parser 106
Figure 7.1: Sprint 3 Gantt chart 119

Figure 7.2: Sprint 3 package diagram 120
Figure 7.3: Sprint 3 class diagram for simviewer.jni 120
Figure 7.4: Sprint 3 class diagram for simviewer.data 121
Figure 7.5: Sprint 3 class diagram for simviewer.visualize 122
Figure 7.6: Sprint 3 class diagram for simviewer.gui part 1 123
Figure 7.7: Sprint 3 class diagram for simviewer.gui part 2 124
Figure 7.8: Sprint 3 class diagram for simviewer.gui part 3 125
Figure 7.9: Sprint 3 class diagram for simviewer.chart.data..... 126
Figure 7.10: Sprint 3 class diagram for simviewer.log 126
Figure 8.1: Sprint 4 Gantt chart 141
Figure 8.2: Sprint 4 class diagram for simviewer.jni 142
Figure 8.3: Sprint 4 class diagram for simviewer.data 143
Figure 8.4: Sprint 4 class diagram for simviewer.visualize 144
Figure 8.5: Sprint 4 class diagram for simviewer.gui part 1 145
Figure 8.6: Sprint 4 class diagram for simviewer.gui part 2 146
Figure 8.7: Sprint 4 class diagram for simviewer.gui part 3 147
Figure 8.8: Sprint 4 class diagram for simviewer.chart.data..... 148
Figure 10.1: Startup dialog guide 162
Figure 10.2: Main window guide 164
Figure B.1: Preliminary study time distribution B-2
Figure B.2: Sprint 1 time distribution..... B-3
Figure B.3: Sprint 2 time distribution..... B-4
Figure B.4: Sprint 3 time distribution..... B-5
Figure B.5: Sprint 4 time distribution..... B-7
Figure B.6: After work time distribution B-8
Figure C.1: Project structure C-3
Figure C.2: XPlanner main page C-4
Figure C.3: Main page for the project..... C-5

Figure C.4: Story information page.....C-6

Summary

Background

SINTEF Fisheries and Aquaculture, our customer, has knowledge and broad competence concerning utilization of renewable marine resources. Their institute contributes to solutions along the whole value chain – from biological and marine production, aquaculture and fisheries to processing and distribution.

One of their projects is fhSim, a simulation engine aimed at the marine industry, accompanied by fhVis, which is a visualization engine for fhSim.

The results of simulations from fhSim are of interest to SINTEF's clients. Currently, to see these results the clients need to perform the actual simulations on their own system, which is a demanding operation.

The customer wishes to be able to give the results of a simulation to their clients which can then be played back in an application better suited for this purpose. This application should support some basic features for analyzing the simulation during playback.

Objectives

Create a standalone tool using the already existing fhVis library for viewing previously computed simulation, and if possible, extend the functionality of this tool to make it useful for statistical analysis and evaluation.

The customer views this project as a way to prove the concept of having output from fhVis in a Java GUI, and is therefore not expecting a full

featured program that can be delivered directly to their clients. Instead they expect a prototype that demonstrates how this can be accomplished.

Results

The project team have researched the relevant technologies and developed a prototype based on the results of this research. The prototype proves that the combination of fhVis and Java is possible, although a lot of problems were discovered in the process.

We ended up with an application that supports playback of simulations using the fhVis library, controlling the playback and viewpoint, and has functionality for analyzing simulations by producing charts.

The customer representatives stated their satisfaction of the progress and results during the review meetings.

Chapter 1

Project Directive

1.1. Chapter introduction

This chapter is the project directive for the student project SimViewer in the subject TDT4290 (Customer Driven Project) for SINTEF Fisheries and Aquaculture.

The documentation related to the group, customer and assignment is given here.

The project directive has the following content:

- Project mandate – This section describes details of the project.

1.2. Project mandate

The project mandate contains information about the project such as name, customer, goals and restrictions.

1.2.1. Project name

The project name was decided by the customer, and is SINTEF SimViewer, or short, just SimViewer. Both the short and long version will be used in this report.

1.2.2. Customer

The customer for this project is SINTEF Fisheries and Aquaculture.

1.2.3. Project background

SINTEF Fisheries and Aquaculture has a simulation technology aimed at marine industry that can perform full simulation of equipment such as trawlers and rigs. This technology is named fhSim for the simulation part, and fhVis which can also show a 3D visualization of the simulation in real time.

The data calculated during the simulation is written to a comma separated value file, which can then be analyzed in retrospect.

To analyze this, SINTEF Fisheries and Aquaculture wishes to be able to playback the simulation's visualization in a standalone tool.

This tool is the SINTEF SimViewer, and a working version is the final result of this project.

1.2.4. Involved parties

The involved parties, including team members, customer representatives and supervisors are found in Appendix A.

1.2.5. Result goals

- R1 – Determine the solution to use the fhVis engine in a Java application.
- R2 – Deliver a functional standalone application that fulfills the requirements specified in Chapter 4.

1.2.6. Project scope

The main goal of this project is to create a standalone tool for playback and analysis of previously performed simulations.

SINTEF Fisheries and Aquaculture wishes this tool to make the current technology in fhSim and fhVis more available for clients in the marine industry.

1.2.7. External conditions

This section describes the external conditions and resources available to this project.

1.2.7.1. People

- The customer. Communication will be through assigned contact persons through e-mail, phone and meetings. Details of customer representatives can be found in A.2.
- The supervisors. Through weekly meetings and e-mail, the supervisors will provide help and guidance throughout this project's lifetime. Details about supervisors can be found in A.3.

1.2.7.2. Computers

The computers available for the team are listed below.

- Personal computers. Every team member has their personal laptop which will be their primary work station. Some also has stationary computers at home for secondary use.
- Group assigned computer. We have an assigned stationary computer available in the team office in P-15.
- The university's computer labs are open for use if needed.

1.2.7.3. Other resources

The following resources are also available for the team.

- Team office. Room 513 in P-15 is available as an office, shared with one other group in Customer Driven Project.
- Team mailing list. kpro7@idi.ntnu.no is setup for team communication.
- Copy machine at IDI offices.

1.2.8. Economy

The estimated workload is 20 hours per student per week. As our group consists of six students, the total workload throughout the duration of the project is approximately 1500 hours.

1.2.9. Duration

The project starts at 26th of August 2008 and continues until the final presentation at 20th of November 2008.

Chapter 2

Project Management

2.1. Chapter introduction

This chapter describes the management of the project team, including time management, team structure, risks and quality assurance.

The purpose of this chapter is to provide the needed information for administrating the project. As changes may occur during the project lifetime, this should be considered a dynamic document subject to continuous updates.

The project management chapter has the following content.

- About Scrum – This section gives an introduction about the Scrum development model.
- Project plan – This section describes the details of the overall project plan. This includes development model and time assignment.
- Project team – This section describes the details surrounding the structure of the team and assigned roles and responsibilities.
- Risk analysis – This section contains the table of risks that was considered for the project.
- Quality assurance – This section describes routines which shall be followed to assure the quality of the project's progress.

2.2. About Scrum

Scrum is an iterative incremental process of software development. It is used in agile software development and was intended for management

of software development projects. The Scrum model consists of a set of predefined roles and practices. The first phase of Scrum (startup phase) is an introductory phase to clarify the scope and management of the project. A project backlog is established, which functions as a prioritized requirement specification for the project.

In Scrum one has iterations called sprints. The most important thing of each sprint is the sprint backlog, which is a prioritized list of requirements chosen at the start of the sprint from the project backlog.

The goal of Scrum is to make teams able to deliver high quality software that fulfills the real need of the customer within the specified time frame and costs.

Sprints

In scrum the iterations are called sprints. Sprints have a Scrum master which is responsible for solving any difficulties arising during the sprint, and the Scrum team which are the developers. Every sprint starts with a meeting where the sprint backlog is created. It consists of entries from the project backlog that are selected in prioritized order.

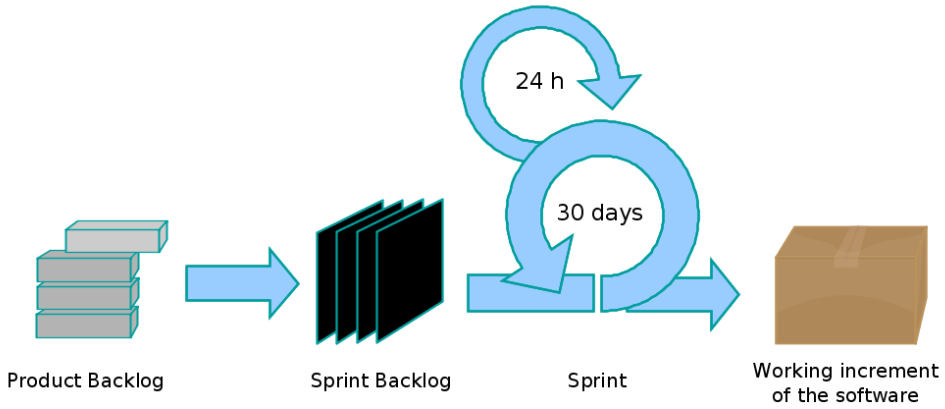


Figure 2.1: Scrum process[1]

[1] Image-source: http://en.wikipedia.org/wiki/Image:Scrum_process.svg GNU Free Documentation License

A sprint usually last between 15 and 30 days. On each working day one is intended to have a sprint meeting to discuss what have been done the last day and what problems did arise. This is a good way to give each member a feeling of responsibility with respect to the rest of the team. It's also a great opportunity to monitor the progress of the project.

The sprint ends with a meeting where the sprint progress and resulting version of the application are evaluated. The customer may during this meeting make any desired changes to the project backlog to accommodate for unpredicted events. This is where the flexibility and agility of Scrum is expressed.

After work

The after work phase takes place after the last sprint, usually lasting only a day or two.

2.2.1. Why Scrum

Scrum is an agile software development model which allows you to do flexible changes during the project progress. Our team decided to use Scrum because we didn't have all the required information to make a total specification of the project beforehand.

A lot of possible issues were reviewed, which could appear during the project progress. Some possible issues are outlined below:

- JNI communication problem
- Problems with the 3D window and displaying it in a Java GUI
- fhVis function support (missing features necessary for implementation of some functions).
- fhVis documentation – no complete documentation, only a document automatically generated from C++ classes

During the iterative process we can find these problems in an early phase of development, and make an agreement with customer how to change some requirements or how to handle these critical issues. If we used the classic waterfall model, we could have problems after the

design phase. Some unsolvable issue could arise, and it would be time consuming to change the whole project design.

Also, the continuous feedback provided at the end of each sprint, will verify and make sure that the team is progressing in the right direction. This will reduce the tension during the development process, as there will be minimal chance for a sudden realization at the end that the project has steered away from the desired result.

2.3. Project plan

This part describes the planning of our project. It contains the main structure of the project plan, how we divided the plan into iterations (sprints), and what we are going to do during these.

This part contains only an overview of what should be done during the iterations. For each iteration there is a chapter containing a detailed description of it.

At the end of this part there is a complete plan with the number of hours spent in the iterations, what tasks we have done and how much time they took.

2.3.1. Structure of the project plan

We have decided to follow the Scrum model for software development which gives us a main idea of how to start dividing the project into phases. The project is divided into 6 parts (iterations):

2.3.1.1. Preliminary studies

During this part we want to prepare the main background for the project development. It contains the following activities:

- Making a requirement specification

  Write a document with the main requirements of the customer. It should go through the process of clarifying information by comments from customer and from us. Final version must be accepted by both sides. See Chapter 4.

- Risk management

  This document contains all the risk factors which can cause our project to fail. It also provides suggestions on how to handle these risk factors. For more information see section 2.5.

- Setting up a project management tool

  For better planning and resource allocation it's important to set up the planning tool, where everyone can write down their spent hours, and what they were used for. Our team decided to use the XPlanner tool. For more info see section 3.3.3.

- Design project report structure

  We need to set the main structure of final report document, complete chapters and sections structure and write main part of pre-study chapter. We need to complete this till September 29th, 2008.

- Technological preliminary study

  In a technological preliminary study it is necessary to find solutions for the main issues. This includes what we need to know before we can start designing and programming the final application. We need to know if the following options are available:

  - JNI communication with C code from SINTEF (fhVis engine)
  - Run 3D scene window initialized by fhVis in Java
  - Use Ogre4j to control OGRE scene initialized by fhVis
  - SWT window with an OGRE scene in Java Swing

2.3.1.1.a Time plan of this part:

Start:	2008-08-26
End:	2008-09-17
Days of work:	17
Estimated working hours:	340h

Expectations at the end:

- Clear requirement specification
- Use cases for the most difficult requirements from our requirement specification
- Created the project backlog
- Finalize the project report structure
- Knowledge about our technical problems

More information about the time plan is included in section B.1. For detailed description of how we managed the preliminary studies, see Chapter 3.

2.3.1.2. Sprint 1

The initial functionality of the SimViewer application will be implemented. The focus will be on the functionality with high priorities in the project backlog. At the end, the first functional version of the application will be presented.

We plan to present the initial GUI design with a basic simulation running. The progress of the first sprint can tell us how we should plan the next iterations.

2.3.1.3. Sprint 2-4

These iterations are basically about adding functionality to the final application. At the end of each sprint we present a new version of the running application.

We also plan to make a presentation for the customer during the last days of each sprint. This should give us feedback from the customer on what we need to change in the next sprint, and what we can consider a fulfilled requirement.

This information will provide background for planning the next sprint.

2.3.1.4. After work

This last iteration should give us time for completing the project report. It will provide a time reserve for some testing and fine tuning of the final application. This will also be where the project report is prepared for print.

2.3.2. Workload

Our team has six members and we are supposed to spend approximately 1500 hours on this project. This part describes how we want to assign this time to the iterations. See the detailed distribution in Table 2.1.

Each iteration will be structured as a small waterfall cycle, which will have their own planning sections. The following table describes the amount of time assigned to each iteration. More detailed description about the hours spent on each iteration is included in the respective chapters.

Iteration name	Days of work	Assigned hours	Actual hours	% of project time
Preliminary study	17	360	312	23.08
Sprint 1	12	300		19.23
Sprint 2	10	240		15.38
Sprint 3	10	240		15.38
Sprint 4	10	240		15.38
After work	3	80		5.13
Total	**62**	**1460**		**93.59**

Table 2.1: Time spent in iterations

Days of work contains only working days and not weekends and holidays.

We cover only 94% of our time budget, but the last 6% is kept as a time reserve for unexpected things that could happen.

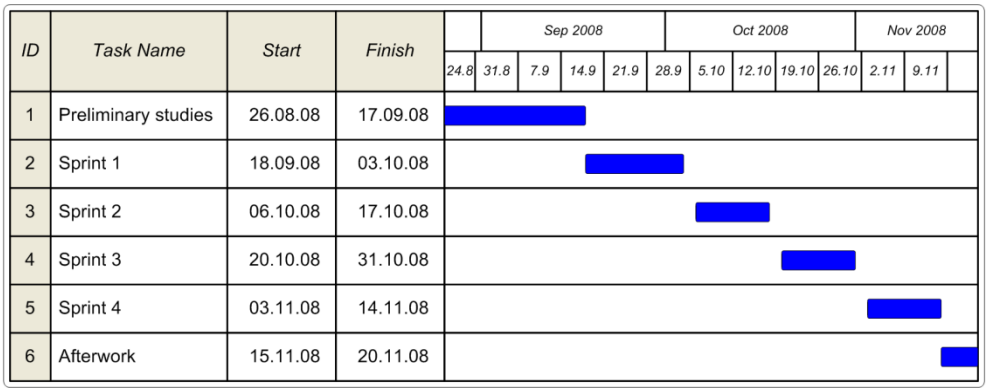
2.3.3. *Milestones and deadlines*

Main deadlines are listed in Table 2.2. Basically most of them are connected with the endings of iterations. The description of each deadline describes our expectations of what we have accomplished.

Name	Deadline	Description
Preliminary study	17.09.2008	Prepared for development start, prepared the project report for first supervisor evaluation.
Project report First delivery	29.09.2008	Finalized the project report for first delivery (changes according to supervisors' comments).
Sprint 1	03.10.2008	First running application with basic functionality
Sprint 2	17.10.2008	Application ver. 2 new functionality
Sprint 3	31.10.2008	Application ver. 3 new functionality
Gold code	11.11.2008	Finished changes in program code. Time for testing; no functional changes allowed.
Sprint 4	14.11.2008	Final tested version of application containing all necessary requirements
After work	19.11.2008	Finished the project report and product documentation. Prepared for final delivery of the project report.
Project report Final delivery	20.11.2008	Delivery of printed project report to IDI information desk IT122.

Table 2.2: Deadlines

2.3.4. Gantt diagrams



ID	Task Name	Start	Finish
1	Preliminary studies	26.08.08	17.09.08
2	Sprint 1	18.09.08	03.10.08
3	Sprint 2	06.10.08	17.10.08
4	Sprint 3	20.10.08	31.10.08
5	Sprint 4	03.11.08	14.11.08
6	Afterwork	15.11.08	20.11.08

Figure 2.2: Overall Gantt diagram

Detailed Gantt diagrams are found in Appendix B.

2.3.5. Progress report from XPlanner

The exported results from XPlanner can be found in section B.1. These describe the progress of the project as well as the actual time distribution for each task.

2.4. Project team

This section describes the organization structure of the team.

Even though we have chosen Scrum as our development model, emphasizing loose and informal team structure, our adaption of the model required some more defined roles, which are distributed as follows:

- Project manager — Jiří Kadlec
- Test manager — Pavel Fatrdla
- Document manager — Jørgen Nystad

- Customer contact Erik Hidle
- Supervisor contact Erling Børresen
- Meeting minutes manager Erik Hidle

2.4.1. *Project manager*

The project manager has the overall responsibility for the project. He must make sure that all active parts of the project are progressing as planned and that all project resources are utilized efficiently.

If any conflicts should arise within the time, the project manager shall assist in resolving these conflicts.

2.4.2. *Test manager*

The test manager is responsible for making sure that the application is thoroughly tested and functioning according to the requirements. This is a task that should be performed during each sprint, and not only near the end of the project.

2.4.3. *Document manager*

The document manager is responsible for accessible documentation for all team members at all times. Also, he is responsible for a uniform document structure throughout, and primarily the formatting and assembly of the project report.

2.4.4. *Customer contact*

The customer contact is responsible for all communication between the customer and team. This primarily concerns written communication, but also to keep track of agreements made with the customer, and making this information available for the rest of the team.

2.4.5. *Supervisor contact*

The supervisor contact is responsible for all communication between the supervisors and team. This primarily concerns written communication, but also to make sure the agenda for meetings is being kept, and that all issues are addressed.

2.4.6. *Meeting minutes manager*

The meeting minutes manager is responsible for documenting all meetings by creating meeting minutes that are available for all parties.

2.5. Risk analysis

Legend: P – Probability, H – High, M – Medium, L – Low.

#	Activity	Risk factor	Consequences	P	Strategy and action	Deadline	Responsible
1	All	Conflicts within the group	**H** Inefficiency Bad communication Unable to meet deadlines	**L**	Arrange team building activities	Continuous	Project manager
2	All	Illness within the group	**H** Inefficiency Unable to meet deadlines Unable to complete tasks Loss of key competence	**M**	Everyone in the group has to make sure to take their vitamins and get proper rest, if illness occurs tasks will be redistributed. Everyone try to get involved in different parts of the project	Continuous	Project manager All

3	All	Communication issues within the group	**M** Inefficiency Redundant work Unable to complete tasks Unable to meet deadlines Erroneous work	**M**	Write meeting minutes, write work distribution plans and follow progress plans	Continuous	Project manager
4	All	Communication issues with customer	**H** Misunderstood tasks Futile project Delays	**L**	Write clear requirement specification and confer with customer, write meeting resumes within 24 hours after customer meetings and make sure customer approves minutes	Continuous	Customer contact Secretary
5	All	Communication issues with supervisor	**M** Misunderstood guidelines	**M**	Write meeting minutes within 24 hours after advisory meetings, make sure to clarify uncertainties	Continuous	Meeting minutes manager Supervisor contact

6	Develop-ment	Faulty planning	**H** Faulty distribution of working hours Unable to complete the project according to speci-fication Unable to meet deadlines Unable to use ideal tech-nologies	**M**	Do thorough preliminary studies Examine alternative technologies	During planning phase	Project manager
7	Finishing	Miss deadline for project report printing	**H** Nothing to deliver	**L**	Book copying machine at least 14 days ahead of the copying at IDI information desk room 122 in It-building.	05.11.2008	Report responsible
8	All	Loss of data	**H** Loss of work Unable to meet deadlines	**M**	Use SVN at IDI to check in code, and use eRoom for all project documents.	Continuous	All
9	All	Conflict with other courses	**M** Unable to meet as a whole group	**M**	Schedule weekly meetings and reserve open time slots ahead.	Continuous	All
10	Develop-ment	Different coding conventions	**L** Some group members and customer unable to understand the code	**M**	Use coding standards	Develop-ment	Program-mers

11	All	Lack of commitment to the project	**M** Unable to meet deadlines Loss of key competence Bad morale	**M**	Motivate team members with team building activities Use the office at P-15 to work together Inform the supervisor of team members that desist from work	Continuous	Project manager
12	All	Task is too complicated	**H** Unable to complete project according to specification Unhappy customer	**L**	Thorough preliminary studies Assign right tasks to the right team member	During planning phase	Project manager
13	All	Incomplete test data	**H** Program that do not work properly with undis-covered errors Unexpected results due to bugs in system	**M**	Request a complete (as possible) test set from customer	Before testing phase	Customer contact
14	Design phase and develop-ment	Bad design decisions	**H** Waste of time Unable to meet deadlines	**L**	Thorough preliminary studies	In the beginning of each sprint	Project manager

15	Prelimi-nary studies	Bad documen-tation / no documentation at all	**H** Unable to understand their system	**H**	Thorough investigation and experimentation with their code. Try to discuss the problems with their chief programmer (Karl Johan)	During preliminary studies	All

Table 2.3: Risk analysis

2.6. Quality assurance

To assure the quality of this project, we have decided to follow the following conventions.

2.6.1. Communication

2.6.1.1. Customer

- All customer communication in written form will be done by the customer contact.
- Customer will be able to answer questions via e-mail at least once per working day.
- All customer communication will be sent in copies to the rest of the team.
- Resumes from customer meetings will be available both for the customer and all team members.

2.6.1.2. Supervisors

- All supervisor communication in written form will be performed by the supervisor contact.
- Relevant documents and agenda for each meeting will be made available at least 24 hours prior to the meeting.

2.6.1.3. Team

- The mailing list kpro7@idi.ntnu.no will be used to communicate relevant information to the whole team.
- All communication shall be done in English.
- Notice of relevant documents and agenda for meetings will be made available at least 24 hours prior to the meeting.

2.6.2. Documents

- All documents shall be stored in the eRoom (see section 3.3.2) provided by the customer.
- eRoom provides version control which shall be used on all documents subject to continuous updates.

Chapter 3

Preliminary Studies

3.1. Chapter introduction

The purpose of the preliminary studies was to study and resolve some of the concerns that arose during the early stages of this project.

One of them is the possibility to use Java and JNI together with the already existing library, fhVis, and access the 3D engine OGRE in a meaningful matter. This will lead to a conclusion of which technologies should be used, as well as giving the group and reader a better understanding of the project and its obstacles.

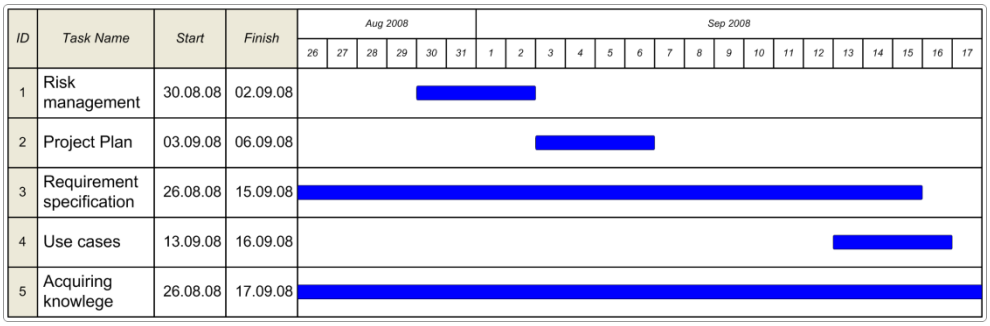
Another was the selection of an easily usable framework for displaying and manipulating charts.

During this phase some management tools were researched to decide on a practical setup for logging of hours spent, version control and document sharing.

The preliminary studies chapter has the following content:

- Resource management – This section describes the tools we researched and chose for managing this project.
- Connecting fhVis with SimViewer – This section describes the research to determine a good way to communicate between fhVis and SimViewer.
- Graphical user interface – This section describes the research to determine the possibilities to have a combination of SWT, AWT and Swing in the same application.
- Charting – This section describes the search for a charting framework covering our needs for SimViewer.

3.2. Gantt diagram



ID	Task Name	Start	Finish
1	Risk management	30.08.08	02.09.08
2	Project Plan	03.09.08	06.09.08
3	Requirement specification	26.08.08	15.09.08
4	Use cases	13.09.08	16.09.08
5	Acquiring knowlege	26.08.08	17.09.08

3.3. Resource management

The following tools for project management have been researched during the preliminary studies.

3.3.1. Version control system

We have chosen to use Subversion as our version control system. It is a CVS like system, but expands the features of CVS quite a bit. The solution we have chosen is provided by IDI and gives us access to the SVN repository through the HTTP protocol.

As the solution from IDI was delayed, we tried to set up our own SVN repository. The easiest way was to use SVN over the SSH protocol, but that did not work out perfectly for the group members running Microsoft Windows. We were about to set up SVNserve when the solution from IDI became available and we chose to use their solution instead.

The use of a distributed version control system was not discussed widely because all group members were familiar with SVN and we found it to suit our needs. For more information about SVN, see their homepage[1].

3.3.2. *Document sharing*

During the first customer meeting the customer representative introduced us to an internal SINTEF tool called eRoom which is used extensively by SINTEF employees for sharing documents and discussing the progress of projects.

SINTEF created an eRoom available for our project. This was an easy way to share documents both internally in the group (with version control) as well as with the customer.

Since the usability of eRoom was deemed to fulfill our project's needs as well as the fact that our customer would prefer to share and handle documents in this way, we decided to take advantage of the offered eRoom, and use this as our primary document repository.

3.3.3. *XPlanner*

This section describes the tool used for time management of our project. It describes why this tool was chosen, what the benefits of this tool were for our team and the main functions of this software. More detailed information can be found in Appendix C.1.

3.3.3.1. Overview

XPlanner is a project planning tool designed for agile methods of development such as eXtreme Programming, Scrum, Test-driven development and so on. It provides functionality that allows control of the project flow by defining iterations, user stories, and tasks (described in C.1.1).

[1] http://subversion.tigris.org

The customer can define stories (functionality of program) which can be divided into multiple tasks with time estimates. After a task is finished the actual hours spent are recorded. This information will help when estimating the next tasks.

3.3.3.2. Conclusion and benefits for team

XPlanner is a project planning tool with a user friendly interface that supports agile method of software development. It's easy to learn and easy to use, but we found a lack of support for exporting time sheets.

Advantages

- Easy to use, easy to learn
- Supports Scrum and other agile methods of development
- Server side application
- Open-source
- Known by one person of the team

Disadvantages

- Missing better export capability
- Project support activities – you can't create tasks without specifying an iteration which makes it difficult to have tasks for project support activities.
- Weak support for milestones and release planning – there is almost no support for this but in Scrum something should be released after each sprint so the end date of each sprint can function as release date in addition to a story for presentation.

Our team decided to use this as a main tool to help us achieve good results and good time planning. Every team member will log the time spent on each tasks with the help of this tool.

3.4. Connecting fhVis with SimViewer

3.4.1. JNI – Creating a connection

FhVis is a C/C++ library contained in a DLL[2]. All the functionality to setup and animate the scene is already provided through this DLL.

Native code, which is used in fhVis, is specific for the operating system, and cannot be accessed from Java without the use of additional components.

SimViewer will be written in Java, and to access the fhVis functionality, we need to create a bridge between SimViewer and fhVis. To create this bridge we decided to use JNI[3].

Java Native Interface (JNI) is a programming framework that provides the possibility to call native functions from Java, and Java functions from native code. This is done by specifying native functions in Java (with the keyword native) and creating a matching C header file specifying the C declarations of the mapped functions.

The functions will then be implemented in C or C++, compiled into a DLL, which must then be loaded in Java.

We created a functioning demo of this during the preliminary studies, to make sure that this was a possible way to bridge SimViewer and fhVis. The demo was a success, and we were able to initialize the fhVis with its engine and perform a simple simulation.

3.4.2. OGRE

FhVis uses OGRE[4] for rendering the simulation scenes.

[2] http://en.wikipedia.org/wiki/Dynamic-link_library
[3] http://en.wikipedia.org/wiki/Java_Native_Interface
[4] http://www.ogre3d.org

It was established during the requirement specification phase that we needed to manipulate OGRE from SimViewer. Since SimViewer will be written in Java, to access OGRE from SimViewer we need a bridge in the same way as for the fhVis library.

One alternative was to manually create the JNI bindings for all the needed OGRE functionality, but as this would become a time consuming and tedious task as the list of required bindings grew, we needed a better alternative.

3.4.2.1. Ogre4j

Ogre4j is a fully developed library providing all the JNI bindings to access all the functionality of OGRE (1). Ogre4j is open-source and licensed under GNU LGPL[5] which does not interfere with the requirements from the customer, so Ogre4j is a good candidate for our project.

We needed to make sure that Ogre4j could actually be used. Concerns included the fact that the C side of Ogre4j needed to be integrated with the DLL containing the JNI bindings to fhVis. This was to ensure that OGRE would not be instantiated twice (once through our DLL and once through the Ogre4j DLL), so that we could work on the fhVis generated scene from our SimViewer application.

Due to the fact that Ogre4j is open-source, we could import all the C source files into our library, and compile it all together into our own DLL containing both our fhVis JNI bindings as well as the Ogre4j bindings.

Another concern was that Ogre4j expects its C side of the JNI bindings to reside in the DLL with the name ogre4j.dll, which would not be the name of our DLL. Again, due to the fact that Ogre4j is open-source we are able to recompile the Java-side of Ogre4j with a new hardcoded name of the expected DLL, which of course matches our own.

[5] http://www.gnu.org/licenses/lgpl.html

3.5. Graphical user interface

Issues regarding the integration of the OGRE render output with our application's GUI needed to be resolved to be able to establish whether some of the requirements were even possible to fulfill.

3.5.1. AWT (Abstract Window Toolkit)

AWT is the first Java graphical user interface (GUI) toolkit. It's part of Java Foundation Classes (JFC). AWT was based on the underlying native system components. For many reasons this was largely replaced by Swing, although the AWT classes are still supported.

There are still some AWT classes (not including components) that are commonly used. Also Java Micro Edition is based on AWT.

3.5.2. SWT (Standard Widget Toolkit)

SWT is an open source GUI library developed by IBM as a part of the Eclipse platform. The purpose of SWT is to provide a common API for accessing native widgets across a spectrum of platforms (2).

The design strategy of SWT was focused on building a fast, simple, portable, essential library that would produce GUI applications closely coupled to the native environment (native look and feel). This is also the main difference between AWT and SWT.

Despite the fact that SWT is written in Java, it is unique for each platform since each of these implementations use the Java Native Interface (JNI) to access the underlying native implementation.

3.5.3. Swing

Swing was built on top of the AWT architecture to provide more sophisticated set of GUI components. All the GUI components are 100%

Java, which means that there is no native code wrapping native GUI components (3).

Swing provides a native look and feel (L&F) that emulates the L&F of other platforms. Among the new features we can find is pluggable L&F that allows application to use the L&F from other platforms.

3.5.4. AWT vs. Swing

The basic difference is that AWT has heavy weight components against Swing's light weight components. Basically, heavy weight means that when we are using AWT components the native code used for getting the view of component is generated by the operating system. This is also the reason why the L&F differs between different operating systems.

On the other hand, in Swing it's the responsibility of Java Virtual Machine (JVM) to generate the view for the components. So the resulting visual appearance is the same on all operating systems where JVM is running.

3.5.5. SWT vs. Swing

There is written a lot of information discussing advantages and disadvantages of either widget toolkit.

One of the most significant debates is about performance. Some people hinted that SWT's dependences on JNI would make it slower because of communication between Java and GUI components, but faster rendering when the data communication is over. However, benchmarks show no clear winner between SWT and Swing (4).

Swing is portable by design, but SWT portability is not a big problem since SWT versions have been implemented for all major operating systems.

SWT has been designed with efficiency in mind. Instead of automatic garbage collection by the JVM, as is the case with Swing, the

programmer has to take care of operating system resources and its de-allocation (5). Wrong usage of this feature of disposing objects can lead to unpredictable results when another object tries to access an already disposed object.

3.5.6. Bridge between SWT and AWT/Swing

SWT contains classes that provide a bridge between SWT and AWT(6). This makes it possible to embed AWT components in SWT and vice versa. However it's not recommended to mix SWT and AWT components unless there is no better choice.

Following is a list of issues that must be considered when using the bridge:

- Multiple event threads – Each toolkit has its own event queue, and each queue should run in a separate thread.
- Modal dialogs – Since SWT and AWT are running in different threads, the SWT event thread has to be explicitly disabled during AWT modal window running and vice versa.
- Synchronizing with system settings – Changes in system environment like font etc. are reflected in SWT components, but not in AWT components. This difference in one application can look very unattractive.
- Reducing flicker – Using of the bridge causes excessive flicker in embedded AWT frame while the window is resized. To avoid this issue one must take additional countermeasures.
- Tab traversal – Problem with tab key behavior when traversing between components from different toolkits. This problem can be solved by additional work on the AWT side of the bridge.

3.6. Charting

3.6.1. Requirements

The following list describes the requirements for the charting framework that had to be fulfilled:

- It should be free; open source license that doesn't interfere with the requirements from our customer
- It should be easy to use
- It should have a large community
- It should be well known and have been in use for a long time (avoid new and unstable frameworks that has not been thoroughly tested)
- It should be able to chart in 3D
- It should be able to persist generated charts to disk in lossless format, compressed format and vector graphics format
- It should be able to update charts or redraw in real time

3.6.2. Candidates

Following is a list of all frameworks researched as possible candidates.

- JFreeChart(7)
- the JCCKit project (LGPL)(8)
- the QN Plot project (BSD License)(9)
- the OpenChart2 project (LGPL)(10)
- the PtPlot project (UC Berkeley copyright)(11)
- the JRobin project (LGPL)(12)
- the JOpenChart project (LGPL)(13)
- the jCharts project (BSD License)(14)
- the JChart2D project (LGPL)(15)
- the Chart2D project (LGPL)(16)
- the ThunderGraph project (LGPL)(17)
- the E-Gantt project (Q Public License)(18)

- the MagPlot project (GPL)(19)

Many of the frameworks lacked vital features and could therefore easily be omitted. Therefore, the only documented framework is JFreeChart.

3.6.3. JFreeChart

3.6.3.1. Advantages

- It is distributed under the terms of the GNU Lesser General Public Licence (LGPL), which permits use in proprietary applications.
- Support for many output types, including Swing components, image files (including PNG and JPEG), and vector graphics file formats (including PDF, EPS and SVG)
- A consistent and well-documented API, supporting a wide range of chart types
- JFreeChart discussion forum is well established and contains a lot of postings made by other users
- It has an active and large community

3.6.3.2. Drawbacks

- No 3D plot
- JFreeChart does not directly support real time charting but it includes an event-notification mechanism that ensures that charts are updated whenever the dataset is updated. However, the chart is completely repainted for each update, which limits the "frames per second" rate that you can achieve with JFreeChart. Typically, updating once per second is fine, but updating multiple times per second results in high CPU load.

3.6.3.3. Examples

To show the simplicity of JFreeChart several example charts together with source code is provided.

3.6.3.3.a Bar chart example

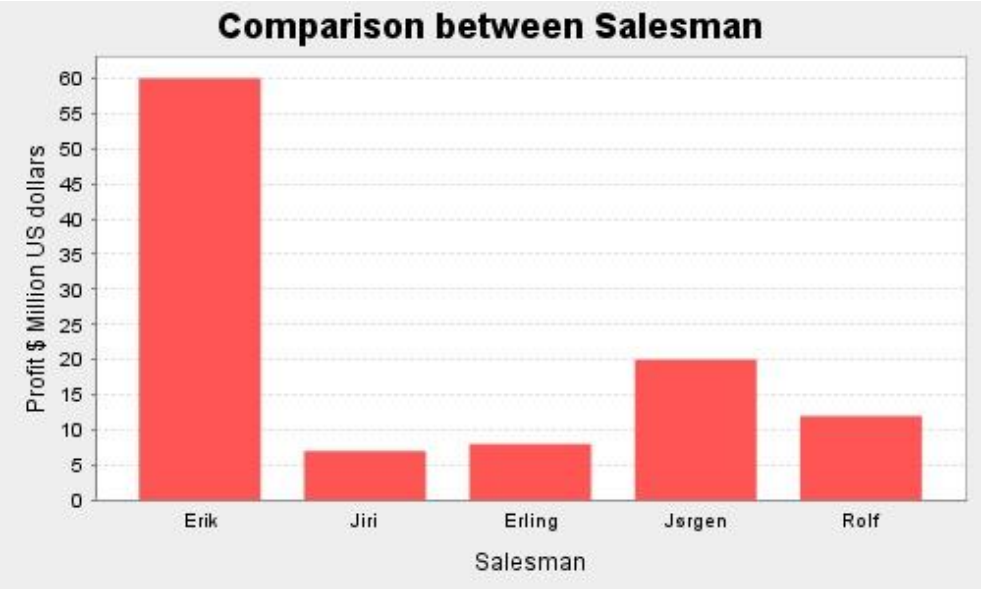


Figure 3.1: Bar chart example output

```
public class BarChartExample {
    public static void main(String[] args) {
        // Create a simple Bar chart
        DefaultCategoryDataset dataset = new DefaultCategoryDataset();
        dataset.setValue(60, "Profit", "Erik");
        dataset.setValue(7, "Profit", "Jiri");
        dataset.setValue(8, "Profit", "Erling");
        dataset.setValue(20, "Profit", "Jørgen");
        dataset.setValue(12, "Profit", "Rolf");
        JFreeChart chart = ChartFactory.createBarChart("Comparison between
                Salesman", "Salesman", "Profit $ Million US dollars",
                dataset, PlotOrientation.VERTICAL, false, true, false
            );
        try {
            ChartUtilities.saveChartAsJPEG(new File("C:chart.jpg"), chart,
                500, 300);
        } catch (IOException e) {
            System.err.println("Problem occurred creating chart.");
        }
    }
}
```

3.6.3.3.b XY chart example

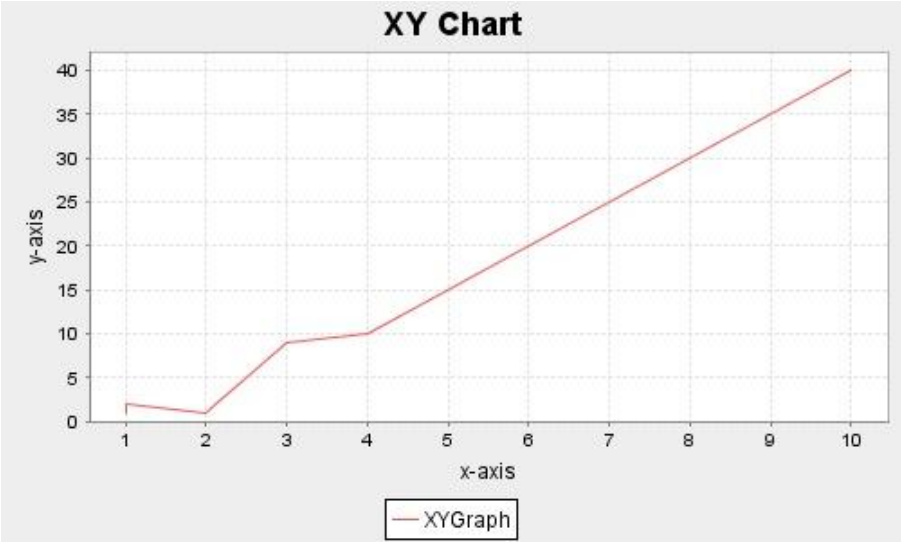


Figure 3.2: XY chart example output

```
public class XYChartExample {
    public static void main(String[] args) {
        //         Create a simple XY chart
        XYSeries series = new XYSeries("XYGraph");
        series.add(1, 1);
        series.add(1, 2);
        series.add(2, 1);
        series.add(3, 9);
        series.add(4, 10);
        series.add(10, 40);
        //         Add the series to your data set
        XYSeriesCollection dataset = new XYSeriesCollection();
        dataset.addSeries(series);
        //         Generate the graph
        JFreeChart chart = ChartFactory.createXYLineChart("XY Chart",
                "x-axis", "y-axis", // Labels
                dataset, // Dataset
                PlotOrientation.VERTICAL, // Plot Orientation
                true, true, // Show legend and use tooltips
                false // Configure chart to generate URLs?
            );
        try {
            ChartUtilities.saveChartAsJPEG(new File("C:chart.jpg"), chart,
                500, 300);
        } catch (IOException e) {
            System.err.println("Problem occurred creating chart.");
        }
    }
}
```

3.6.3.3.c *Pie chart example*

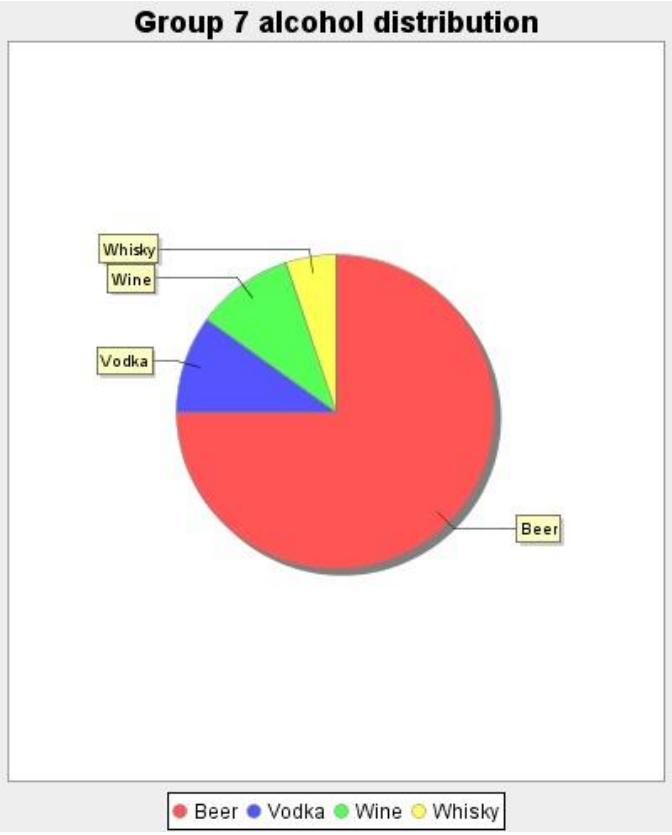


Figure 3.3: Pie chart example output

```
public class PieChartExample {
    public static void main(String[] args) {
        // Create a simple pie chart
        DefaultPieDataset pieDataset = new DefaultPieDataset();
        pieDataset.setValue("Beer", new Integer(75));
        pieDataset.setValue("Vodka", new Integer(10));
        pieDataset.setValue("Wine", new Integer(10));
        pieDataset.setValue("Whisky", new Integer(5));
        JFreeChart chart = ChartFactory.createPieChart(
               "Group 7 alcohol distribution", pieDataset,
                true, true, true
            );
        try {
            ChartUtilities.saveChartAsJPEG(new File("C:chart.jpg"), chart,
                400, 500);
        } catch (Exception e) {
            System.out.println("Problem occurred creating chart.");
        }
    }
}
```

Chapter 4

Requirements Specification

4.1. Chapter introduction

This chapter deals with the requirements specification of the SimViewer, and will be updated throughout the project lifetime.

The purpose of this chapter is to establish and thoroughly document the requirements for the project, in an unambiguous and definite manner. This will be used to establish a common understanding of the project's goals and restrictions between the project team and SINTEF Fisheries and Aquaculture.

This document has been approved by the customer and should be used as a first source to answer any questions regarding the SimViewer application.

The requirements specification has the following content:

- Overview – A general description of some expectations for SimViewer
- Use cases – Visual and textual use cases for use in the requirement specification process.
- Requirements – The actual formal requirements for SimViewer. In case of any conflict, these take precedence.
- Project backlog – The Scrum project backlog for use when planning each sprint.

4.2. Overview

The purpose of this project is to develop a tool for visual analysis of time domain simulations. This tool shall be able to read simulation results from fhSim/fhVis simulations and present the results in a standalone application for interactive analysis.

4.2.1. Proposed interaction with fhVis

It is the intention that the SimViewer application shall make use of two input files:

- The simulation output file containing positions, forces, etc. of the system
- The simulation configuration file which contains the setup data for the simulation

The SimViewer application will read the simulation configuration file and pass it to fhVis directly as an xml string. fhVis will then parse the configuration data and set up a 3D scene by use of OGRE based on the initial state values defined in the configuration. A handle to this scene will be accessible from the SimViewer application, providing all necessary access to the functionality in OGRE.

The simulation output file contains the system states (positions etc.) for each time step. It also contains output from each simulation object. Updated values for the system states may be sent to the fhVis engine, and a new rendering of the 3D scene may be performed through an existing function call.

Interactive analysis of the scene shall be performed by i.e. changing the camera position, selecting visual objects for investigating their properties and other actions. It is also the intention that functionality for more traditional data analysis shall be available in the SimViewer application, i.e. time series charts and simple statistics for information found in the simulation output file.

4.2.2. *User characteristics*

The typical user of the planned SimViewer has the following characteristics:

Client in the marine industry without extensive technical knowledge, but with detailed knowledge about the simulation

4.2.3. *Dependencies*

SimViewer will be dependent on the already developed code base of fhSim/fhVis, provided by the customer.

SimViewer is required to use the open source 3D engine OGRE for the 3D visualization part of the tool, and the existing integration between OGRE and fhVis.

4.3. Use cases

This section describes some use cases describing the user interaction with SimViewer.

4.3.1. Textual use cases

UC1 Set input files

Participating actors:	User
Flow of events:	1. The user can choose to load input files from the menu. 2. The user inputs the address of the configuration file and simulation output file.
Entry condition:	Program running in normal state
Exit condition:	Ready to read simulation output file, or error due to inconsistency in file
Quality requirements:	None

UC2 Start scene visualization with objects

Participating actors:	User
Flow of events:	1. Uses case starts when user click on start visualization button. 2. Program check files existence 3. IF EXIST Then 3.1 Program loads files and creates the scene. 3.2 If there is a failure in consistency of XML configuration file, it will report an error 4. ELSE Reports an error
Entry condition:	Program running in normal state
Exit condition:	Visualization running
Quality requirements:	None

UC3 Change visualization speed

Participating actors:	User
Flow of events:	1. User chooses a different speed by using the speed control. 2. The program changes visualization speed according to selected speed
Entry condition:	Program running in visualization state
Exit condition:	Visualization running, with different speed than before
Quality requirements:	None

UC4 Change camera position

Participating actors:	User
Flow of events:	1. User presses arrow keys in order to change camera position 2. Program moves with a camera according to key direction
Entry condition:	Program running in visualization state
Exit condition:	Visualization running, with different camera position than before
Quality requirements:	None

UC5 Attach camera to object

Participating actors:	User
Flow of events:	1. User chooses attach camera via the control panel. 2. SimViewer sets the change internally and updates camera position on the scene.
Entry condition:	Program running in visualization state, and one object is selected
Exit condition:	Visualization running, with different camera setting than before.
Quality requirements:	None

UC6 Select object(s) from object list

Participating actors:	User
Flow of events:	1. User selects objects from the list of all objects list. 1.1. In order to select multiple objects, the user must press and hold the CTRL key while selecting. 2. SimViewer highlights these objects and sets their state to "selected" in order to know which objects to process for object specific functions later
Entry condition:	Program running in visualization state
Exit condition:	Visualization running, with new highlights, and new selected settings on objects.
Quality requirements:	None

UC7 View object's properties

Participating actors:	User
Flow of events:	1. Program will display properties of selected objects in main window.
Entry condition:	Program running in visualization state and some object is selected
Exit condition:	Program displaying properties of selected object
Quality requirements:	None

UC8 Take screenshot of the scene

Participating actors:	User
Flow of events:	1. User can choose to take screenshot via an action in the control panel. 2. SimViewer prompts for where to save file 3. SimViewer saves the file to a specified location.
Entry condition:	Program running in visualization state
Exit condition:	Program running in Visualization state
Quality requirements:	None

UC9 Generate charts of object's properties

Participating actors:	User
Flow of events:	1. User selects which property of selected object he wants to display in chart. 2. SimViewer creates new window with appropriate chart.
Entry condition:	Program running in visualization state and at least one object is selected.
Exit condition:	Visualization running and application creates a new window with chart.
Quality requirements:	None

UC10 Generate charts with function among objects' properties

Participating actors:	User
Flow of events:	1. User selects function among objects' properties whose he wants to display in chart. 2. SimViewer creates a new window with the appropriate chart.
Entry condition:	Program running in visualization state and at least two objects are selected.
Exit condition:	Visualization running and application creates new window with chart.
Quality requirements:	None

UC11 Save charts

Participating actors:	User
Flow of events:	1. Use case starts when user want to save a chart 2. User select save chart from a popup menu in the chart window. 3. Program display menu with save dialog 4. User chooses name and path of file. 5. Program saves the file.
Entry condition:	Program running in visualization state
Exit condition:	Program running in Visualization state
Quality requirements:	None

4.3.2. *Use case diagram*

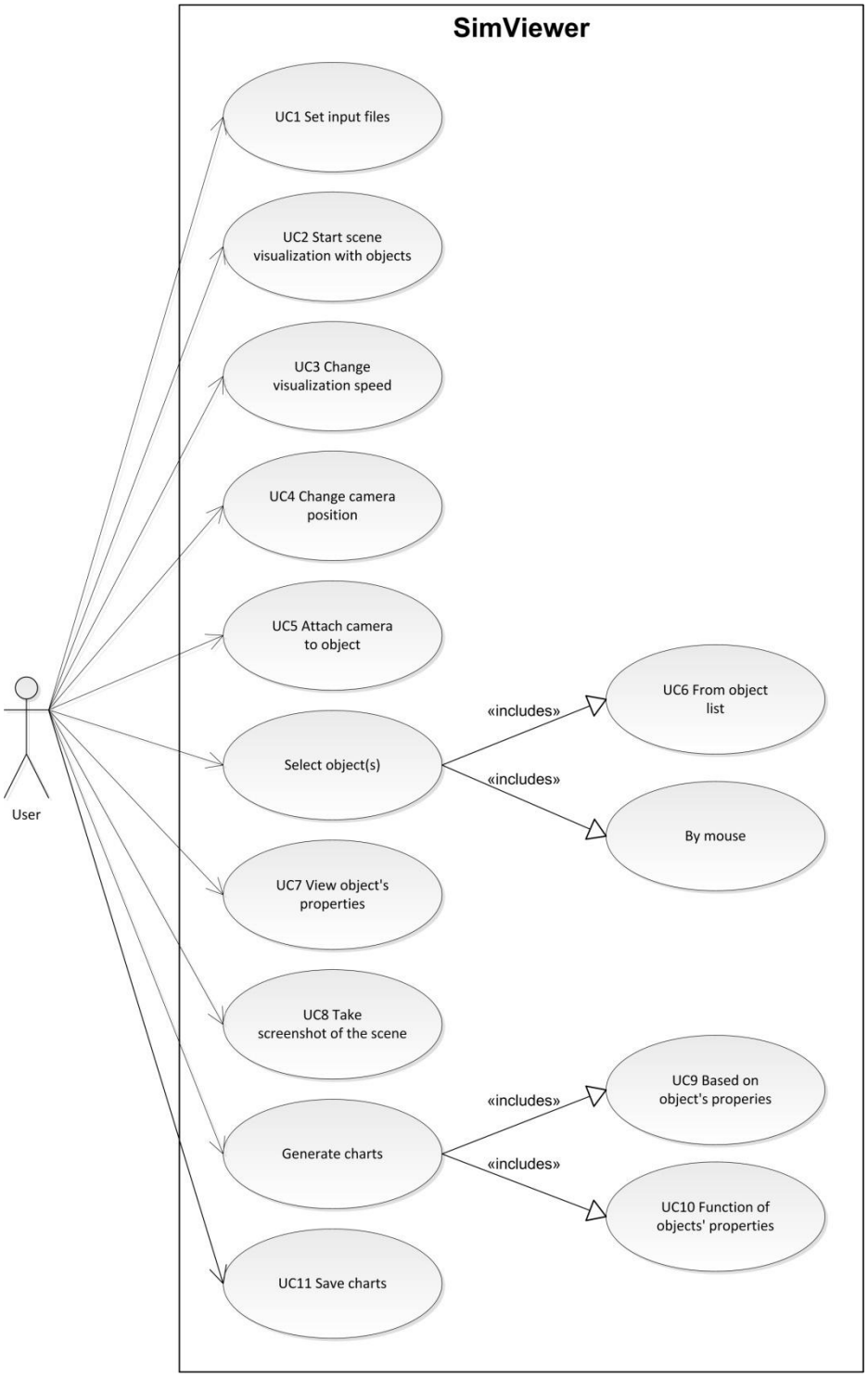


Figure 4.1: Use case diagram

4.4. Requirements

This chapter contains the complete descriptions of the requirements for the finished application SimViewer.

4.4.1. Functional requirements

This section contains the functional requirements for the finished application. The requirements have been ordered into priorities of high, medium, low and very low. Some requirements have been marked optional, as the client has defined parts of the functionality as unessential, but wanted. These requirements will be fulfilled if the limited time is sufficient.

4.4.1.1. Reading of simulation results

F1.1 Reading configuration file

Description:	The application will be able to read fhSim configuration file and send it through JNI to fhVis engine and set up the scene. This will be the XML document and there is no need to make any changes to it. If fhVis rejects the file, the application displays an error.
Priority:	High

F1.2 Reading simulation output file

Description:	The program will be able to read simulation output file. Format is CSV and it will contain information about objects in simulation scene and their behavior (forces, positions). There will be one column with time steps. Each row will describe all properties of objects at specified time.
Priority:	High

1.3 Checking consistency

Description:	Program will check the consistency between XML and CSV file. If there is any error the program shows an error message.
Priority:	Optional – very low

4.4.1.2. Interactive 3D scene

F2.1 Showing simulation environment in 3D

Description:	The application will contain the 3D visualization environment of the simulation.
Priority:	High

F2.2 Time stepping

Description:	Application will be able to visualize the simulation in real time and also allow use of real-time factors. Pause/rewind/fast forward-buttons will also be provided. It will also contain a speed indicator (e.g. half speed of real simulation).
Priority:	Medium

F2.3 Camera position

Description:	Application will provide the ability to change camera position and angles, using buttons, keys and/or mouse gestures.
Priority:	High

F2.4 Following camera

Description:	There will be option to position the camera relative to some object in the scene. If the object moves, the camera moves too.
Priority:	Low

F2.5 Several windows / viewpoints

Description:	The application could have more than one frame displaying the 3D environment. It can allow you to see the objects from different angles at the same time.
Priority:	Optional – low

F2.6 Definition of camera trajectories as a function of simulation time

Description:	It will be possible to set trajectory by a function of time and then the camera should follow this trajectory in time.
Priority:	Optional – very low

F2.7 Adjusting scene environment

Description:	This functionality allows you to adjust scene environment (e.g. lighting, skybox, sea surface, seabed, other objects)
Priority:	Optional – low

4.4.1.3. Analyzing simulation objects

F3.1 List of simulation objects

Description:	There shall be a list of all simulation objects.
Priority:	Medium

F3.2 Selecting objects with mouse

Description:	By clicking on a simulation object with the mouse, the system should select and highlight this object.
Priority:	Optional – medium

F3.3 Selecting objects from list

Description:	Selecting the object from a list of all simulation objects. The system should highlight the selected object.
Priority:	Medium

F3.4 Display information of simulation object

Description:	The system shall be able to display basic information about the selected object (e.g. forces and positions). Vectors should be identified and collapsed by default.
Priority:	Medium

F3.5 Display statistical values of selected object(s)

Description:	The system shall be able to display mean values (and std. deviations etc.) of inputs, states and outputs (i.e. position, force, distance and angle relevant to the selected object(s)).
Priority:	Optional

F3.6 Display information of multiple simulation objects

Description:	The system shall be able to display information in the 3D scene about inputs, states and outputs (i.e. forces, positions, distance, angle and strain between the selected objects). Generally, this is to show functions of signals from several objects. One such function is the distance between two objects (abs(p1-p2)). Other functions shall be specified, too.
Priority:	Optional – low

F3.7 Display time plot of force of selected object

Description:	The system shall be able to display a time plot of selected outputs and states.
Priority:	Low

F3.8 Display time plot of functions of signals available in multiple objects

Description:	The system shall be able to chart functions of inputs, states and outputs of objects. Such functions f, may be (v1 and v2 are vector time series of inputs, states or outputs): • f=v2-v1 (Component wise (vector)) • f= abs(v2-v1) (Absolute value (vector) / distance) • f=acos(dot(v1,v2)/(abs(v1)*abs(v2)) (angle between vectors) • v1 and v2 may occur from selecting three objects. The user should be able to specify the time dimension.
Priority:	Low

F3.9 Displaying visual cues between objects to display distance and angle

Description:	The system shall be able to display visual cues that can show distance or angle between 2 or more selected objects.
Priority:	Optional

4.4.1.4. Exporting results

F4.1 Take screenshot during the simulation

Description:	The user shall be able to take a screenshot that is persisted to disk. The user can specify the format of the screenshot (JPEG, PNG etc.)
Priority:	Low

F4.2 Record video of parts of simulation

Description:	The user shall be able to record parts of the simulation by specifying start and end time, the output shall be AVI or MPEG.
Priority:	Optional

F4.3 Save charts to image file

Description:	The user shall be able to save charts to image files. The user can specify the format of the image file (e.g. JPEG, PNG, SVG, etc.)
Priority:	Medium

F4.4 Saving scene environment

Description:	Save the complete state of the program.
Priority:	Optional - low

4.4.1.5. Statistics

F5.1 Filtering

Description:	Possibility to add filters to smooth chart curves to remove noise.
Priority:	Optional - high

F5.2 Fourier transforms

Description:	Doing Fourier transforms of charts.
Priority:	Optional – very low

4.4.2. Non-functional requirements

N1.1 System requirments

Description	The application must be able to run on a standard laptop as of 2008. • 2.0 GHz dual core • 2GB RAM • Dedicated graphics card

N1.2 Startup time

Description	When the simulation input file size is less than 500 MB, the startup time for the application shall not exceed 20 seconds. Running on the specified computer (*N1.1*).

N1.3 Environment

Description	The application should be able to run on Microsoft Windows XP.

N1.4 Performance-based usability

Description	The users of SimViewer (Engineers, ship owner, etc.) should not use more than 10 minutes to learn how to set up and run a simulation in SimViewer changing camera aspects and zoom.

N1.5 Performance-based usability

Description	When a user jumps to another frame out of reach of the current playback buffer, the buffer update should not take more than 2 seconds.

N1.6 Language

Description	All communication with the user shall be in English. That is all text shown to user in the graphical interface must be written in English.

N1.7 Accuracy of interpolation

Description	When interpolation is done, the accuracy should be to a minimum of two places behind the decimal point.

N1.8 Serviceability

Description	The application shall generate a log of each execution. It shall contain each program state it enters: • Exception block entry • Method entry • Etc. This log may be used to debug or isolate faults and monitor application execution.

N2.0 Documentation

Description	The application shall come with a user manual that describes how to install the application, and use all of its futures. The manual shall be readable by anyone, who has had some experience with installing and using applications (basic users).

4.5. Project backlog

Item #	Description (with reference to req. number)	Hours est.	Risk est.
High			
1	1.1 Reading configuration file	5	L
2	1.2 Reading simulation output file, create data structure	60	M
3	Sending simulation data to fhVis	60	M
4	2.1 Showing simulation environment in 3D	60	M
5	Initial GUI construction	30	L
6	Initial GUI functionality - Provide actions for the functions made in the sprint	40	L
7	2.3 Camera position	15	L
Medium			
8	2.2 Time stepping	80	M
9	3.1 List of simulation objects	15	L
10	3.3 Selecting objects from list	15	L
11	3.4 Display information of simulation object	30	L
12	3.7 Display time plot of force of selected object	50	M
13	4.3 Save charts to image file	5	L
Low			
14	2.4 Following camera	30	M
15	3.8 Display time plot of functions of signals available in multiple objects	80	H
16	4.1 Take screenshot during the simulation	15	L
	Total – non-optional requirements	**590**	

Item #	Description (with reference to req. number)	Hours est.	Risk est.
Optional			
17	2.5 Several windows / viewpoints	80	H
18	2.6 Definition of camera trajectories as a function of simulation time	100	H
19	2.7 Adjusting scene environment	50	M
20	3.2 Selecting objects with mouse	50	H
21	3.5 Display statistical values of selected object(s)	55	H
22	3.6 Display information of multiple simulation objects	40	L
23	3.9 Displaying visual cues between objects to display distance and angle	150	H
24	4.2 Record video of parts of simulation	60	H
25	4.4 Saving scene environment	40	L
26	5.1 Filtering	40	M
27	5.2 Fourier transforms	85	H
	Total – all requirements	**1340**	

Table 4.1: Project backlog

Chapter 5
Sprint 1

5.1. Chapter introduction

The purpose of this chapter is to document the progress, results and acquired knowledge during sprint 1 of the project.

What is covered here took place in the time period between September 17th and October 3rd, 2008.

This chapter has the following contents:

- Sprint backlog – The sprint backlog includes the elements from the project backlog where the focus of this sprint was placed.
- Design – Any system design that took place during this sprint.
- Development – Describes the development that took place during this sprint.
- Testing – Describes a set of tests used to make sure that the application functions according to the requirements
- Conclusion – A summary of the progress and problems encountered during this sprint.

5.2. Sprint backlog

Item #	Description (with reference to req. number)	Hours est.	Risk est.	Respon-sible	Assigned members
1	1.1 Reading configuration file	5	L	Jørgen	Jørgen
2	1.2 Reading simulation output file, create data structure	60	M	Rolf	Rolf, Erik and Erling
4	2.1 Showing simulation environment in 3D	60	M	Jiri	Jørgen, Jiri and Pavel
5	Initial GUI construction	30	L	Pavel	Jiri and Pavel

6	Initial GUI functionality - Provide actions for the functions made in the sprint	40	L	Erik	Erik and Erling
	Total – sprint 1	**195**			

Table 5.1: Sprint 1 backlog

5.3. Sprint Gantt chart

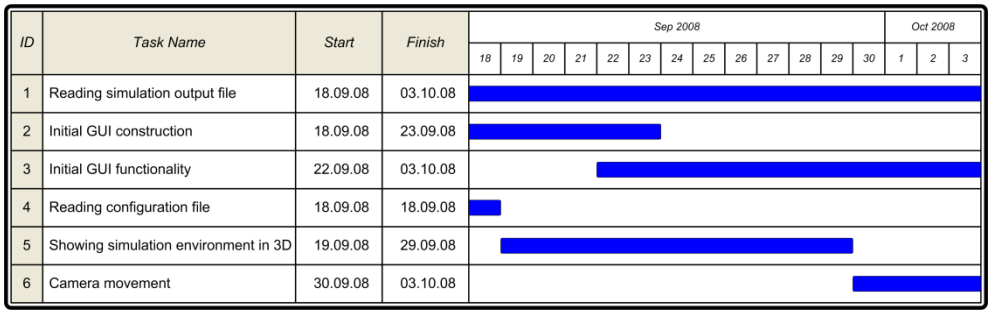

ID
Task Name
Start
Finish
Sep 2008
Oct 2008
18 19 20 21 22 23 24 25 26 27 28 29 30 1 2 3
1 Reading simulation output file 18.09.08 03.10.08
2 Initial GUI construction 18.09.08 23.09.08
3 Initial GUI functionality 22.09.08 03.10.08
4 Reading configuration file 18.09.08 18.09.08
5 Showing simulation environment in 3D 19.09.08 29.09.08
6 Camera movement 30.09.08 03.10.08

Figure 5.1: Sprint 1 Gantt chart

5.4. Design

In this sprint we designed the initial structure of the SimViewer application. The design takes form of diagrams such as class diagram, activity diagram, sequence diagram and other UML diagrams.

These diagrams help to better understand problematic parts of the application. It also provides a good background for developers who would like to continue on this project at a later time.

Because of the Scrum method of development, there is a high chance that any of the diagrams will be changed in later sprints.

5.4.1. *Package and class diagrams*

The package and class diagrams contain the basic structure of the application and the most important classes and packages. There is also a brief textual description of classes and packages.

5.4.1.1. Package diagram

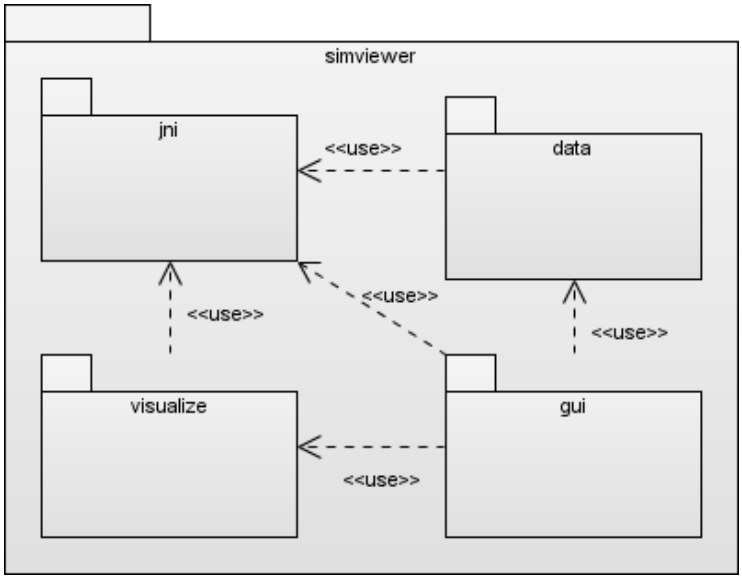


Figure 5.2: Sprint 1 package diagram

5.4.1.2. Class diagram for simviewer.jni

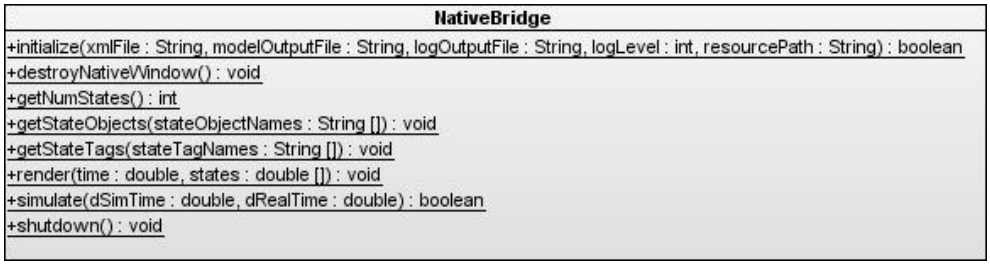

NativeBridge
+initialize(xmlFile : String, modelOutputFile : String, logOutputFile : String, logLevel : int, resourcePath : String) : boolean
+destroyNativeWindow() : void
+getNumStates() : int
+getStateObjects(stateObjectNames : String []) : void
+getStateTags(stateTagNames : String []) : void
+render(time : double, states : double []) : void
+simulate(dSimTime : double, dRealTime : double) : boolean
+shutdown() : void

Figure 5.3: Sprint 1 class diagram for simviewer.jni

5.4.1.3. Class diagram for simviewer.data

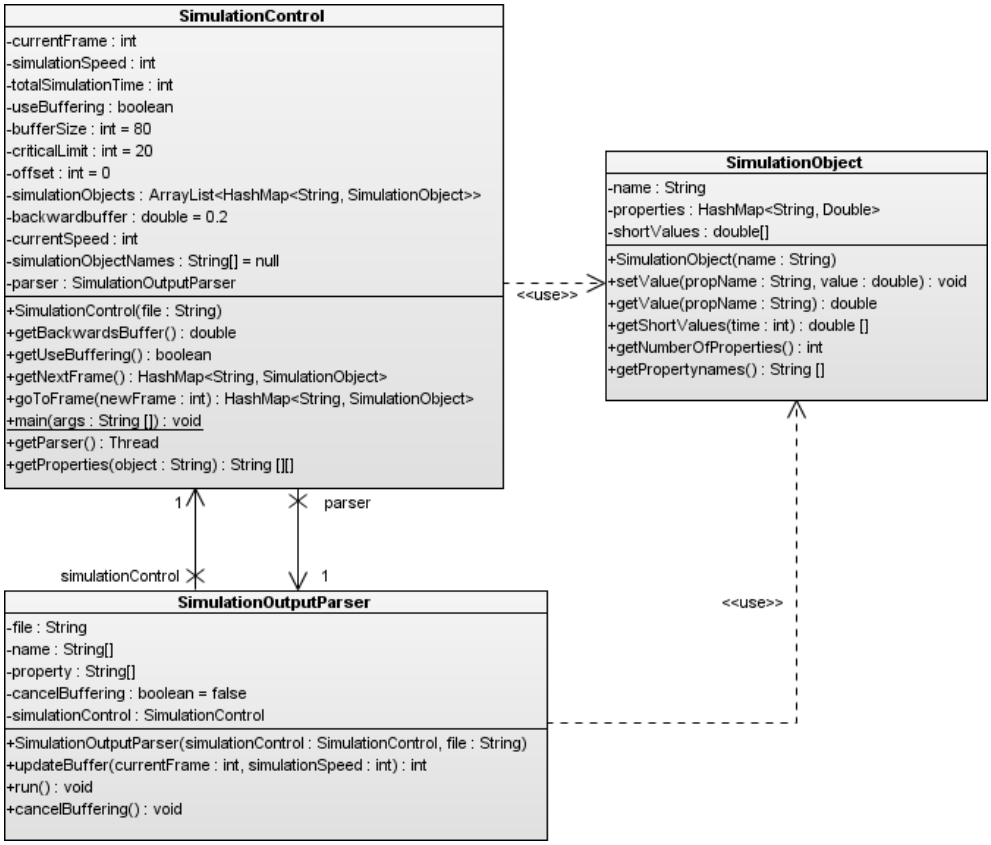


Figure 5.4: Sprint 1 class diagram for simviewer.data

5.4.1.4. Class diagram for simviewer.visualize

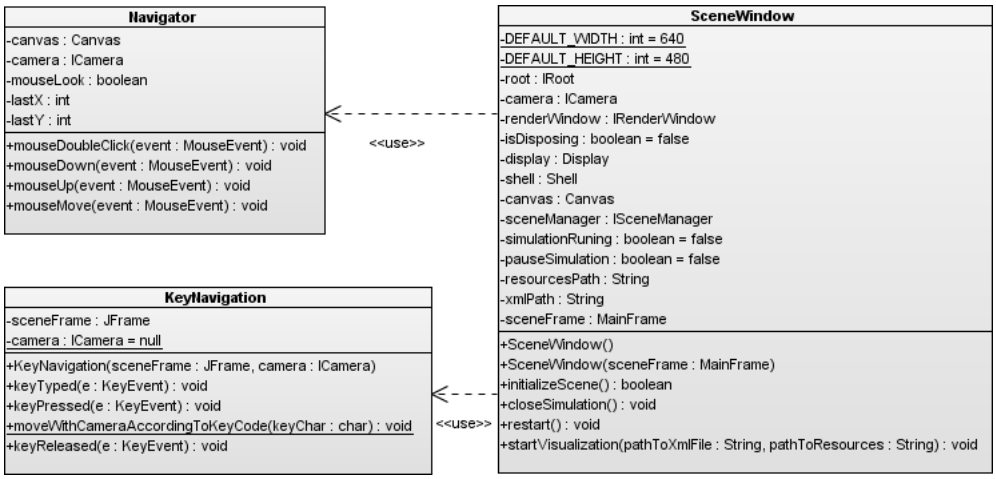


Figure 5.5: Sprint 1 class diagram for simviewer.visualize

5.4.1.5. Class diagram for simviewer.gui part 1

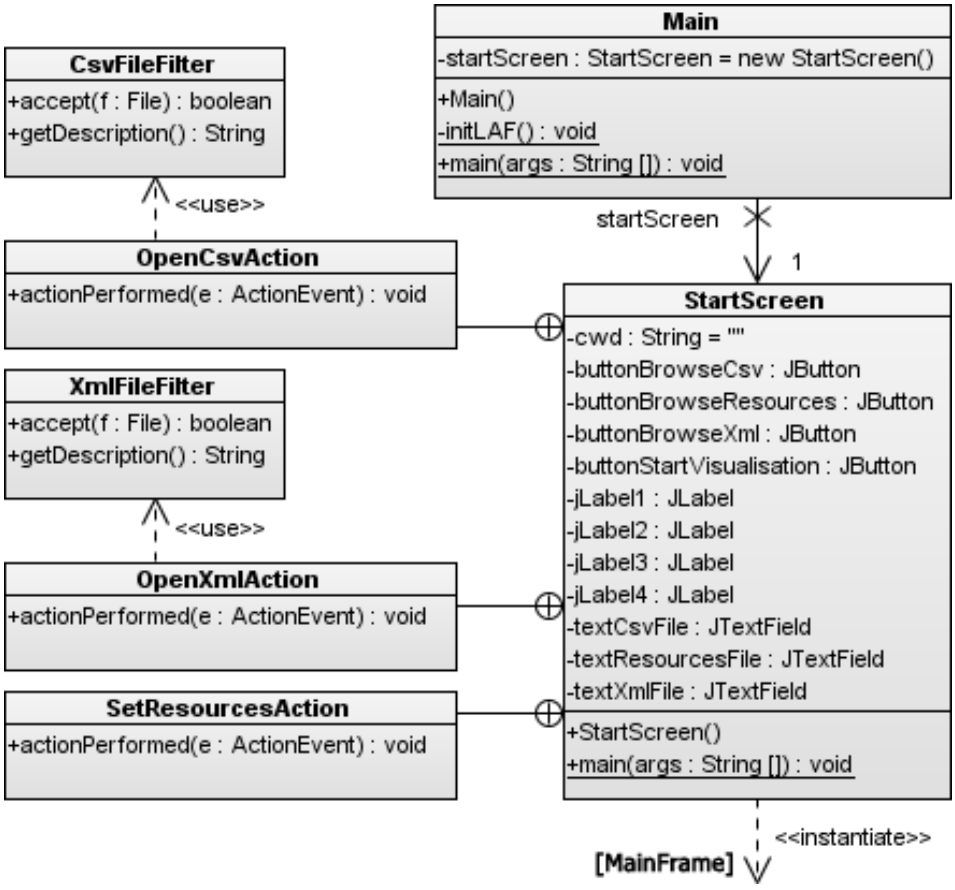


Figure 5.6: Sprint 1 class diagram for simviewer.gui part 1

5.4.1.6. Class diagram for simviewer.gui part 2

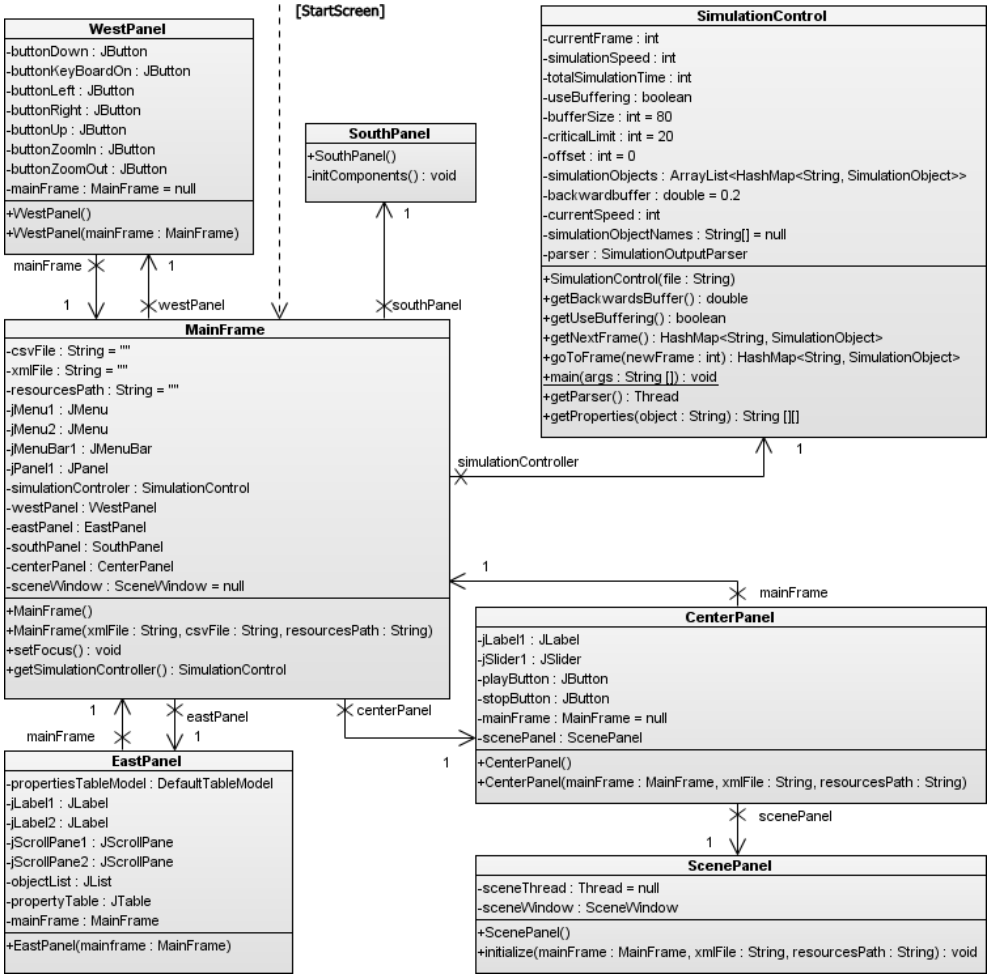


Figure 5.7: Sprint 1 class diagram for simviewer.gui part 2

5.4.1.7. Package descriptions

5.4.1.7.a simviewer

This is the main package which contains the other packages and the Main class which is used to start the application.

5.4.1.7.b simviewer.chart.data

This package contains all functionality of charting in SimViewer. It will contain classes for calculating functions for charts of angle, distance

between objects and absolute values. This package stays empty in sprint 1, due to the priority of the charting feature.

5.4.1.7.c simviewer.chart.gui

This package is also connected with charting in SimViewer, but contains the classes generating the graphical user interface. It's also empty in the first sprint.

5.4.1.7.d simviewer.data

This package contains classes important for processing simulation output file from fhVis (CSV file). It provides functionality like parsing, calculating some interpolations between frames and a good structure for sending state information to fhVis for running simulation playback.

5.4.1.7.e simviewer.gui

This package covers the main graphical user interface of SimViewer. It contains the main frame window and components for displaying controls, properties of objects, playback functions like stop, play and a slider for moving in simulation. It also provides a panel for the scene component which is provided by the visualization package.

5.4.1.7.f simviewer.jni

Classes in this package are managing the connection between our Simviewer application and fhVis simulation/visualization engine. It provides JNI function calls between C++ and Java. These functions are necessary for controlling visualization through fhVis.

5.4.1.7.g simviewer.visualize

This package covers all functions for displaying and controlling scene environment. These functions are using the JNI package for communication with fhVis or directly calls Ogre4j library to communicate with 3D graphics engine.

It contains the component which is able to display a fhVis scene and provides the Java component (java.awt.Canvas) which makes the

simviewer.gui package able to display the scene in the GUI. It also contains navigators for controlling camera by mouse, keyboard or buttons.

5.4.1.8. Class descriptions

This section contains tables of classes with their description. This description for each class should help with understanding the whole diagram.

5.4.1.8.a simviewer

Class name	Description
Main	Starting point of the application. It initializes the StartScreen class and runs the application.

5.4.1.8.b simviewer.data

Class name	Description
SimulationControl	Controls reading of simulation output file provides good structure for reading data by program during visualization playback.
SimulationObject	Represents one simulation object in the CSV file. It contains its properties names.
SimulationOutputParser	Parses the CSV file and puts data about objects in the prepared structure.

5.4.1.8.c simviewer.gui

Class name	Description
CenterPanel	Central panel of the GUI which contains scenePanel for displaying the main scene and time controls of scene such as play, pause, time slider, etc.
CsvFileFilter	This is used as a file filter class and it provides filtering of CSV file for the FileChooser which is used in the StartScreen.
EastPanel	This contains the list of objects and table with objects' properties.

ScenePanel	This is just an extension of JPanel which provides space for scene window component.
SouthPanel	This panel is used as a status bar of the application.
StartScreen	This class is representing the first screen seen by the user when running the application. It contains file choosers for choosing the right input files necessary for starting the simulation playback. Then it provides functionality to start simulation from this window.
WestPanel	WestPanel is used for providing camera control functionality. For now, this is buttons for moving the camera, but the functionality will be increased in later sprints.
XmlFileFilter	Same as CsvFileFilter, but for XML.

5.4.1.8.d simviewer.jni

Class name	Description
NativeBridge	NativeBrigde is the main connection point with the fhVis library. It provides function calls through JNI to fhVis.

5.4.1.8.e simviewer.visualize

Class name	Description
KeyNavigation	Provides keyboard control of camera position. Implements Swing KeyListener.
Navigator	This implements SWT mouse listener used for controlling the camera position.

SceneWindow	SceneWindow extends java.awt.Canvas and provides a component for displaying an OGRE scene initialized through fhVis. It takes care of calling initialization functions starting the SWT loop and preparing SWT components for displaying the scene. Then it connects the SWT and AWT components to be able to use it in Swing GUI. It holds all attributes connected with the scene such as OGRE Root, Camera, RenderWindow and so on.

Table 5.2: Sprint 1 class descriptions

5.5. Development

5.5.1. Development of NativeBridge

This section describes the development process of NativeBridge. This is the part which connects SimViewer and fhSim/fhVis and allows the visualization and simulation engine to be used in Java.

5.5.1.1. Task definition

The purpose of NativeBridge was to expose functions for initialization, simulation, render (which includes altering state), info retrieval and shutdown.

This task includes writing both the C++ and Java side of the JNI bindings.

Also, the Ogre4j library had to be integrated into the same DLL as the NativeBridge, to allow manipulation of fhVis' instance of OGRE.

5.5.1.2. Solution

The following function mapping was created in JNI:

- initialize() – this method takes input parameters such as paths to XML configuration file and output files and passes this to fhVis' initializeFromFile(). This results in a native window (Win32, unlike a Java Swing Frame) with the output of OGRE directed to this window.
- destroyNativeWindow() – this method kills the native window created by fhVis. This was needed to remove the native window after we redirected the OGRE output to our Java application.
- getNumStates() – this method maps directly to getNumStates() in fhVis. This returns the number of states that should be passed to the render() function.
- getStateObjects() – this method takes an empty String array and fills it with the simulation objects' names. The size of the String array must be the same as the number of states returned by getNumStates().
- getStateTags() – this method takes an empty String array and fills it with the simulation object's property names. The size of the String array must be the same as the number of states returned by getNumStates().
- render() – this method takes a time and a double array with states and passes it to the render() function in fhVis. This function will be used to render an updated scene with the states we specify, thus enabling playback from file.
- simulate() – this method performs a simulation step. This function is here just to be able to show some output while the playback functionality is still not available.
- shutdown() – this method destroys the instance of fhVis created in the native DLL, and should release all used resources.

The Java side of the mapping consists only of native function definitions in the class NativeBridge, which was then used to generate a C/C++ header file with the tool javah[7], included in JDK[8].

The C++ side was then created following the structure of the generated header file. This is where the calls to fhVis takes place, as well as conversion between Java types and C++ types.

5.5.1.3. Encountered problems

The development of the mapping was relatively straight forward, with only small issues with type conversion.

There were some issues with initialization of the visualization, where we had to figure out the right way to redirect the OGRE output into SimViewer. The main problem of it all was the cryptic error messages we got when something failed.

The error messages were typically of the form "INTERNAL_ERROR", with a problematic frame reference to something like "kernel32.dll+0x1eb33". This made it very hard to debug, since we often had to guess what was causing the error.

This took a lot of time to figure out, but we managed to make a working solution before the end of this sprint.

However, while all of the functions now worked as they should in Windows XP, some of the functions caused a failure in Windows Vista of the type "ACCESS_VIOLATION". This applied to the functions getStateObjects() and getStateTags(). Solving this problem was postponed to the next sprint.

5.5.2. Development of SceneWindow

This section describes the development process of the SceneWindow component. It is one of the most important components because it

[7] http://java.sun.com/j2se/1.5.0/docs/tooldocs/windows/javah.html
[8] Java Development Kit

provides functionality to see the visualized scene in a Java environment.

5.5.2.1. Task definition

The main goal of this task was to figure out how to run a 3D scene in Java Swing GUI. This includes the following problems:

- Get render output from the 3D engine written in C++ initialized by another C++-library (fhVis initializes the OGRE engine)
- The component initialization procedure initializes 3D scene through NativeBrigde, which does JNI calls to fhVis.
- Provide functionality such as controlling and changing the scene for other classes.
- Provide possibility to run the 3D engine and control the application at the same time.

This task was tightly connected with NativeBrigde for JNI calls, because this component needed to initialize the 3D engine through JNI. This functionality was provided by NativeBridge.

Problems concerning JNI are described in section 5.5.1.

The result of this task would be a component which could be used in Netbeans' designer. This suggested that the component had to extend some Java Swing component and fulfill the definition of a JavaBean.

5.5.2.2. Solution flow of actions in development

This was the one of the main tasks in first sprint, because displaying a simulation is the main purpose of this project.

Development started with analyzing the example application with OGRE render output in the Java GUI. The example used the SWT library for displaying the output in Java and we needed a Java Swing component.

After a few hours we found the first solution on how to run the scene in Java with SWT. Then we started solving the SWT problem. We found a way to put an SWT Canvas into an AWT Canvas which is compatible

with Swing components. This gave us the main idea of what kind of component we needed to extend.

After the introductory research we created the SceneWindow class (see 5.4.1.8 for description). During the design of the class we decided which methods and attributes we needed for this component and created a skeleton of the class. The main operations of the class are listed below:

- initializeScene() – this method creates and shows the 3D scene in the SceneWindow component. It uses NativeBridge to initialize the scene in fhVis as well as creating SWT components with bridges to AWT. At the end initialization this method calls runSimulation().
- runSimulation() starts the simulation of the scene. Our program is not supposed to do simulation, but since the actual playback functionality would be implemented later, the simulation functioned as a test of the initialization and display.
- closeScene() – method is called before the program ends. It cleans up the environment and disposes the 3D engine classes and shut fhVis down.

When we had a skeleton of the component and clarified descriptions of methods, we could start with coding the functionality of each method to fulfill the requirements.

We did not encounter any specific problems during the coding process. When we tried to use the SceneWindow with other GUI components, a lot of problems appeared.

We realized that the SWT loop was incompatible with the Java Swing message dispatcher, which resulted in a full freeze of the Swing parts of the GUI.

We found a possibility to start a new thread for the SWT loop, but this caused new access conflicts between the threads.

We didn't have enough time to analyze all the risks involved by using several threads, but we would be aware of these during later development.

In the final application we will update states from CSV file instead of simulating, which makes it necessary to access attributes from the SWT thread.

5.5.2.3. Conclusion and possible improvements

At the end of the first sprint we delivered a functional version of the SceneWindow component which can provide a Swing component with running 3D scene. This fulfilled the requirements for this task.

Some issues and problems during development are listed below:

- Restarting scene – Because of a lot of issues with this feature, we decided to require a restart of the application to load a new scene. Customer approved this solution.
- Threads issues – we encountered some problems with threads since we need to use one thread for SWT loop and another for running the simulation. This must be considered in later sprints.

According to the agreement with the customer we will remove functionality for restarting scene with a new configuration file.

5.5.3. Simulation controller framework

5.5.3.1. Task definition

A simulation output parser with automatic buffering and playback control, with the following constraints:

- Big file (> 100 MB)
- Number of simulation objects and length of simulation vary
- Limited memory
- Need to be fast and efficient

5.5.3.2. Solution

- Keep only small part of file in memory
- Buffer size vary with file size
- When a critical limit is reached, the buffer is reloaded in a separate thread

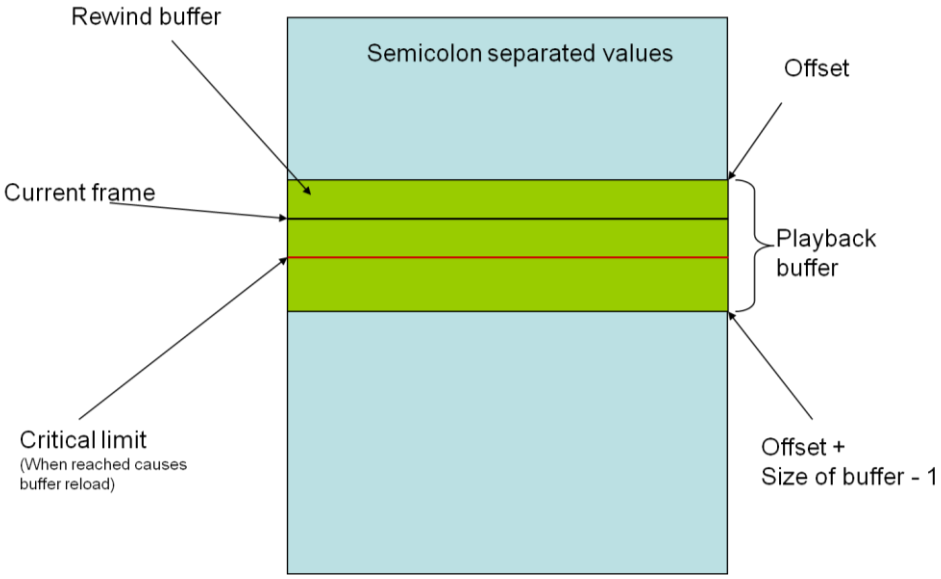


Figure 5.8: Simulation controller buffering

5.5.3.3. Parsing algorithm

Initially the first number of lines (dependant on the buffer size) are read from the file into a data structure that contains the time and the objects with their respective properties. When Current frame reach a critical limit it triggers a buffer update that runs in a separate thread. This allows the playback to continue while the buffer is loaded.

When buffering is completed the new buffer is automatically loaded. Because the user might want to rewind we also make sure to keep a small part of the buffer in the opposite direction of the playback.

The framework contains appropriate methods for playback, jump to frame and methods for accessing a list of simulation objects and their properties and values for each time instant. The users of the framework will only be exposed to these methods.

Buffering and other advanced features are handled automatically and are out of scope for the users of this framework.

5.5.4. *Simulation object list for GUI*

This section explains changes done to make the JList and JTable of the EastPanel work. The JList contains a list of all the current simulation objects and the JTable contains pairs of property names and values for the selected simulation object.

EastPanel now has reference to the MainFrame object which allows EastPanel to access SimulationControl.

Initially for the JList we call appropriate methods of SimulationControl to get a String array with simulation object names. This array is passed through the JList constructor making it show the simulation objects correctly. The simulation objects are also sorted alphabetically in ascending order.

For the JTable we create a DefaultTableModel that holds the values that the JTable displays. Each time a valueChanged event is thrown (resulting from a selection change in the JList) we update the model with help from methods available in SimulationControl to get a two dimensional String array which contains properties and property values for the selected object.

The JList of the EastPanel is now sensible to valueChanged event (called whenever the value of the selection changes) which is handled by a ListSelectionListener which passes the event further to objectListHandler. In objectListHandler the fields of the JTable (propertiesTable) is updated by removing every item of its model and then adding new properties with respective values of the selected object.

5.6. Testing

ID/Title	1.0 Testing of reading CSV files
Description	In this test we want to test whether the functionality for loading the CSV file into an appropriate data structure work as required. The CSV file contains all the simulations objects and property values for the objects. It can be of variable length and the size of this file cause the complexity to be pretty high. It is therefore important to test this functionality on CSV files of different size and with variable number of objects and attribute values. We have an example CSV file from the customer and will use this file to make several different CSV files. The parsing functionality will be tested with 3 CSV files: One file containing of 4 simulation objects and a total of 10 MB One file containing of 8 simulation objects and a total of 100 MB One file containing of 16 simulation objects and a total 500 MB
Precondition	The SimViewer is not running, the CSV file has already been created and placed inside a folder in the working directory.
Post-condition	The output file should be loaded in the data structure and there should not have occurred any exceptions or problematic states that lead to unwanted behavior.
Comments	The parsing seems to work fine, no problems with big files either.
Pass	Ok

ID/Title	2.0 Testing of showing the simulation environment in 3D in Java
Description	To test the simulation environment in 3D a small simulation setup file will be loaded. This simulation is not a real simulation, it does not need a CSV file with objects and property states. It's only to check that we are able to use fhVis to show visualization in a java swing window.
Pre-condition	SimViewer is not running.
Post-condition	SimViewer is running and the simulation is looping. 3 small balls will move around "randomly" in the screen window.
Comments	Works very nice but there is no functionality for resizing the simulation window.
Pass	Ok

ID/Title	3.0 Test of initial GUI – close window
Description	Will test if it possible to close the application window while visualization is running.
Pre-condition	SimViewer is running.
Post-condition	SimViewer is closed and all resources that were being used by the application is released.
Comments	
Pass	Ok

ID/Title	3.1 Test of initial GUI – startup dialog input XML, CSV and resources path
Description	Test that start visualization cannot be performed without specifying paths to XML, CSV and resources path.
Pre-condition	SimViewer has been executed and the dialog pops up, all fields are empty.
Post-condition	SimViewer is running and an error message pops up telling the user that not all of the paths have been specified.
Comments	
Pass	Ok

ID/Title	3.2 Test of file choosers in startup dialog
Description	Test that all the browse buttons work. There is a total of three browse buttons: one for XML, CSV and resources.
Pre-condition	SimViewer has been executed and the startup dialog pops up.
Post-condition	When browse button is pushed a dialog where one can specify a directory or file should pop up.
Comments	
Pass	Ok

5.7. Conclusion

In the first sprint we created the main connection between Java and fhVis, implemented in the NativeBridge class and nativelib library. Thanks to this, we could read and pass a configuration file to fhVis, show 3D simulation in Java and implement basic camera movements.

For playback of the simulation, we also needed to start working on the simulation controller. This framework parses an fhSim output file and creates a structure of objects in simulation and also shows them in a table in the GUI.

After getting some additional information from the customer we discovered that we also had to do interpolation and buffering of an input file and change the GUI structure of the simulation object table. We weren't able to make these changes in sprint 1 so we planned to include them in the next sprint.

Chapter 6
Sprint 2

6.1. Chapter introduction

The purpose of this chapter is to document the progress, results and acquired knowledge during sprint 2 of the project.

What is covered here took place in the time period between October 6th, and October 17th, 2008.

This chapter has the following contents:

- Sprint backlog – Includes the elements where the focus of this sprint was placed.
- Design – Any system design that took place during this sprint.
- Development – Describes the development that took place during this sprint.
- Testing – Describes a set of tests used to make sure that the application functions according to the requirements
- Conclusion – A summary of the progress and problems encountered during this sprint.

6.2. Sprint backlog

Item #	Description (with reference to req. number)	Hours est.	Risk est.	Respon-sible	Assigned members
3	Sending simulation data to fhVis	60	M	Jørgen	Jørgen, Pavel
2	1.2 Reading simulation output file, create data structure	60	M	Rolf	Rolf, Erik and Erling
11	3.4 Display information of simulation object	30	L	Jiri	Pavel and Erling
9	3.1 List of simulation objects	15	L	Jiri	Pavel and Jiri

10	3.3 Selecting objects from list	15	L	Jiri	Pavel and Jiri
	Total – sprint 2	**180**			

Table 6.1: Sprint 2 backlog

6.3. Sprint Gantt chart

ID	Task Name	Start	Finish
1	Sending simulation data to fhVis	06.10.08	17.10.08
2	Reading simulation output file	06.10.08	11.10.08
3	List of simulation objects	06.10.08	10.10.08
4	Selecting objects from list	11.10.08	16.10.08
5	Display information of simulation object	11.10.08	18.10.08

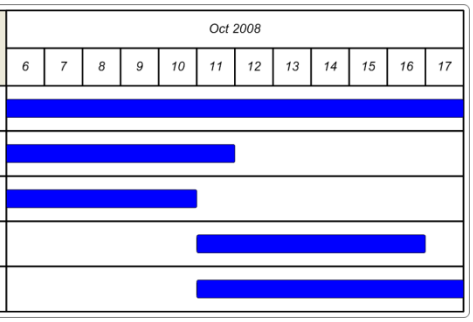


Figure 6.1: Sprint 2 Gantt chart

6.4. Design

6.4.1. Class diagrams

Following are the class diagrams for sprint 2. The package diagram is identical to the one in sprint 1, and can be found in 5.4.1.1.

New classes are marked with blue.

Significantly altered classes are marked with green.

6.4.1.1. Class diagram for simviewer.jni

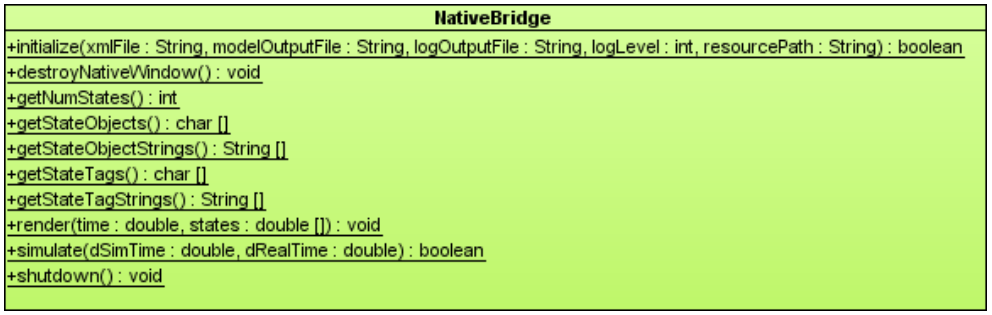

NativeBridge
+initialize(xmlFile : String, modelOutputFile : String, logOutputFile : String, logLevel : int, resourcePath : String) : boolean
+destroyNativeWindow() : void
+getNumStates() : int
+getStateObjects() : char []
+getStateObjectStrings() : String []
+getStateTags() : char []
+getStateTagStrings() : String []
+render(time : double, states : double []) : void
+simulate(dSimTime : double, dRealTime : double) : boolean
+shutdown() : void

Figure 6.2: Sprint 2 class diagram for simviewer.jni

6.4.1.2. Class diagram for simviewer.data

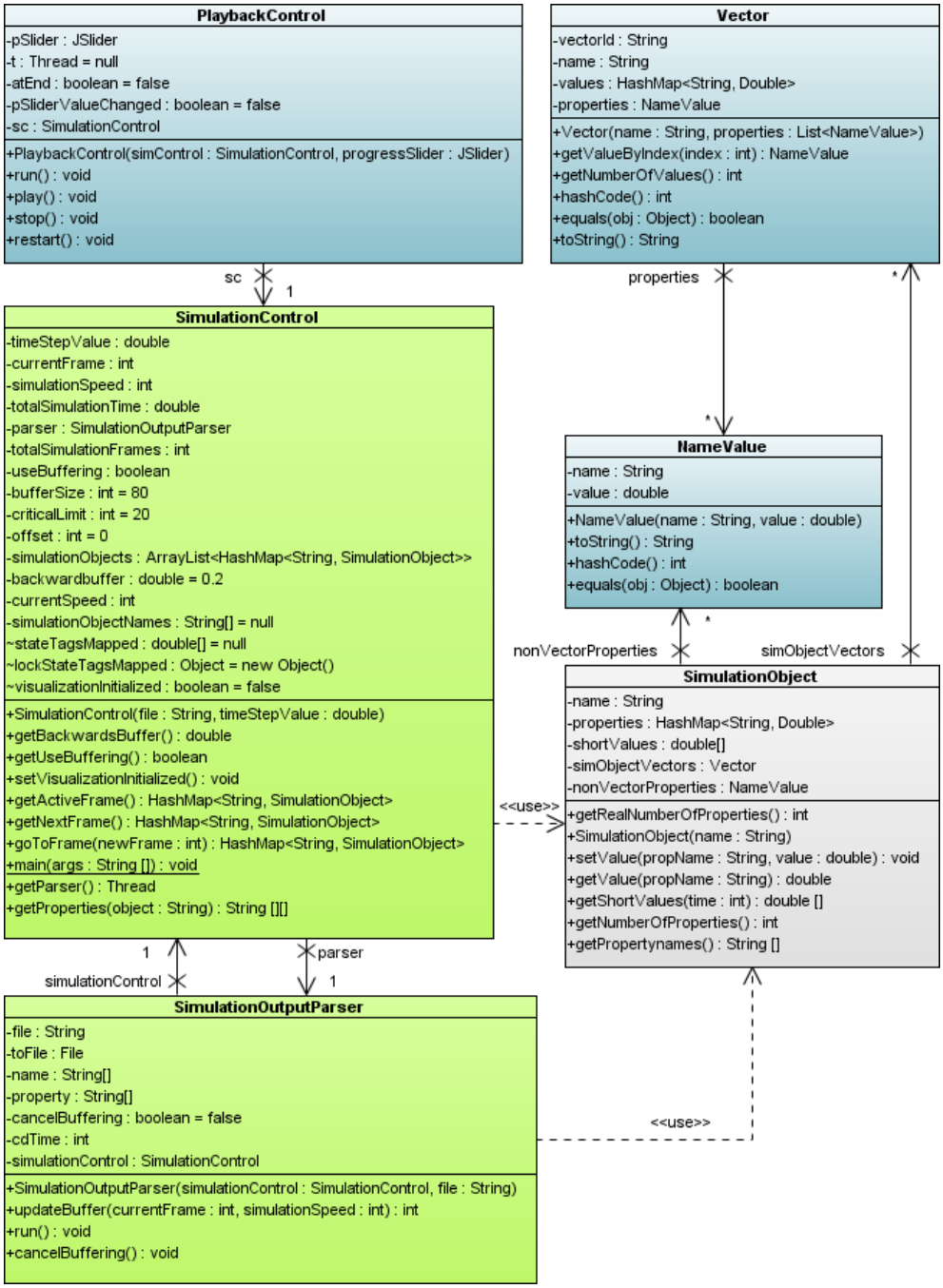


Figure 6.3: Sprint 2 class diagram for simviewer.data

6.4.1.3. Class diagram for simviewer.visualize

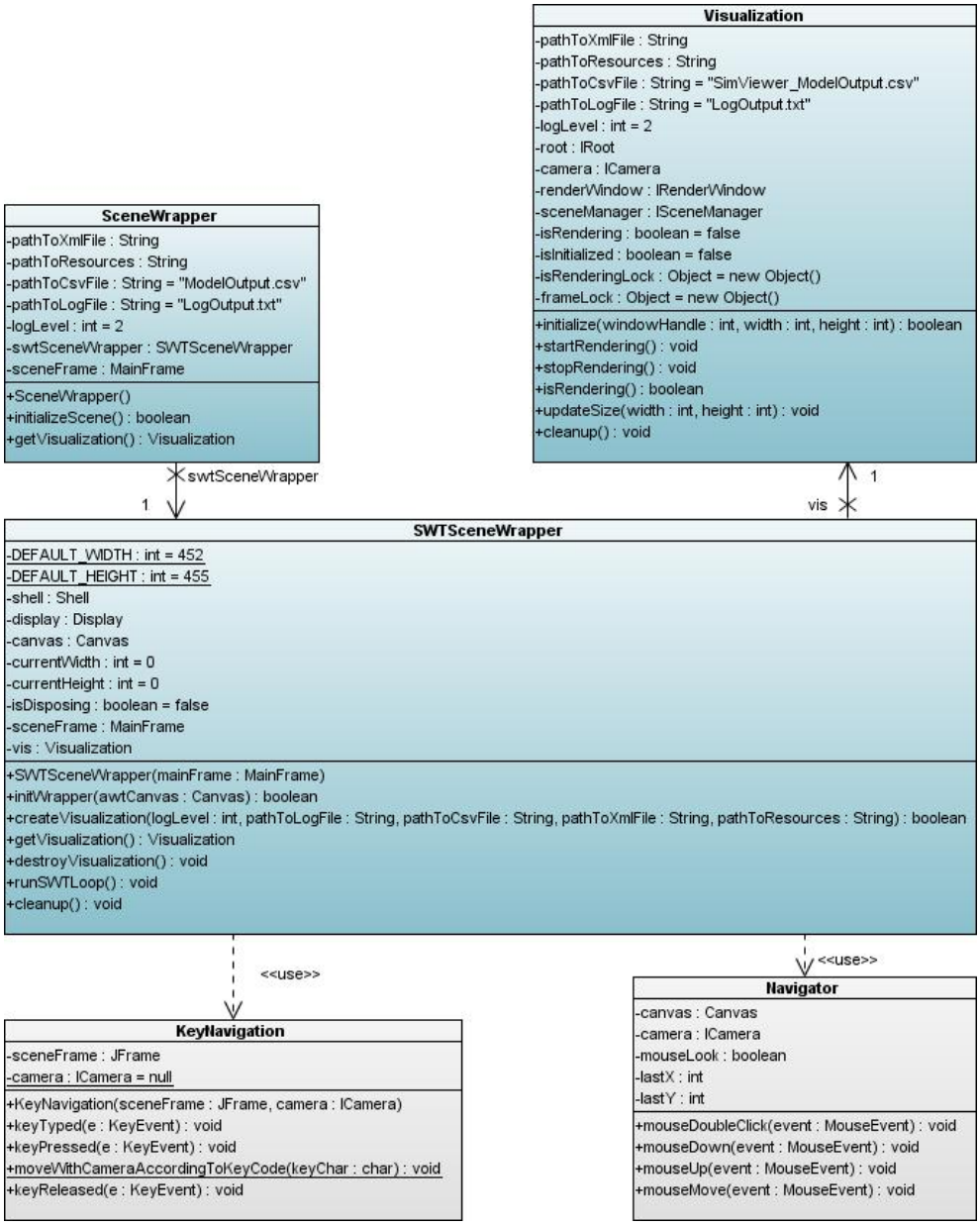


Figure 6.4: Sprint 2 class diagram for simviewer.visualize

6.4.1.4. Class diagram for simviewer.gui part 1

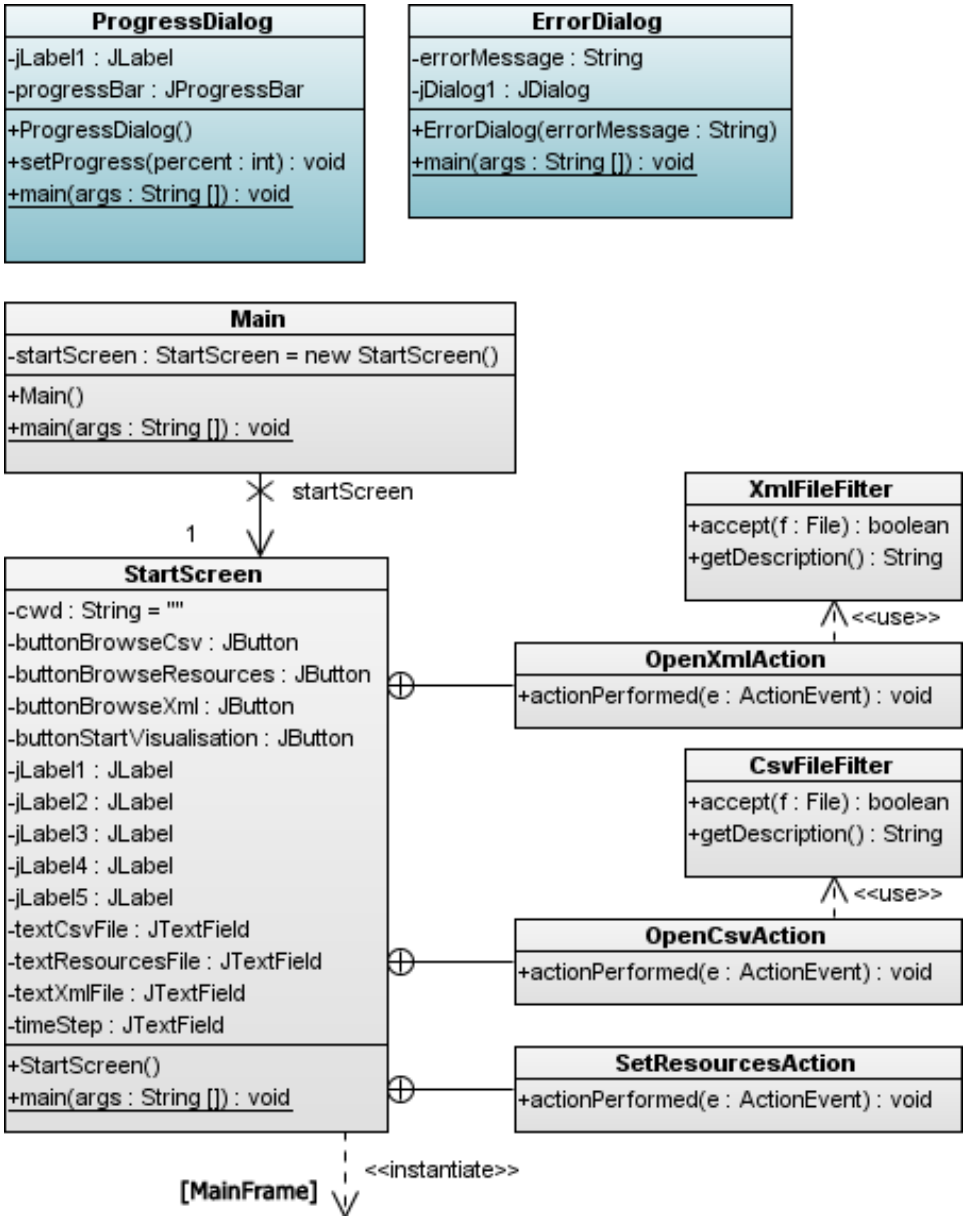


Figure 6.5: Sprint 2 class diagram for simviewer.gui part 1

6.4.1.5. Class diagram for simviewer.gui part 2

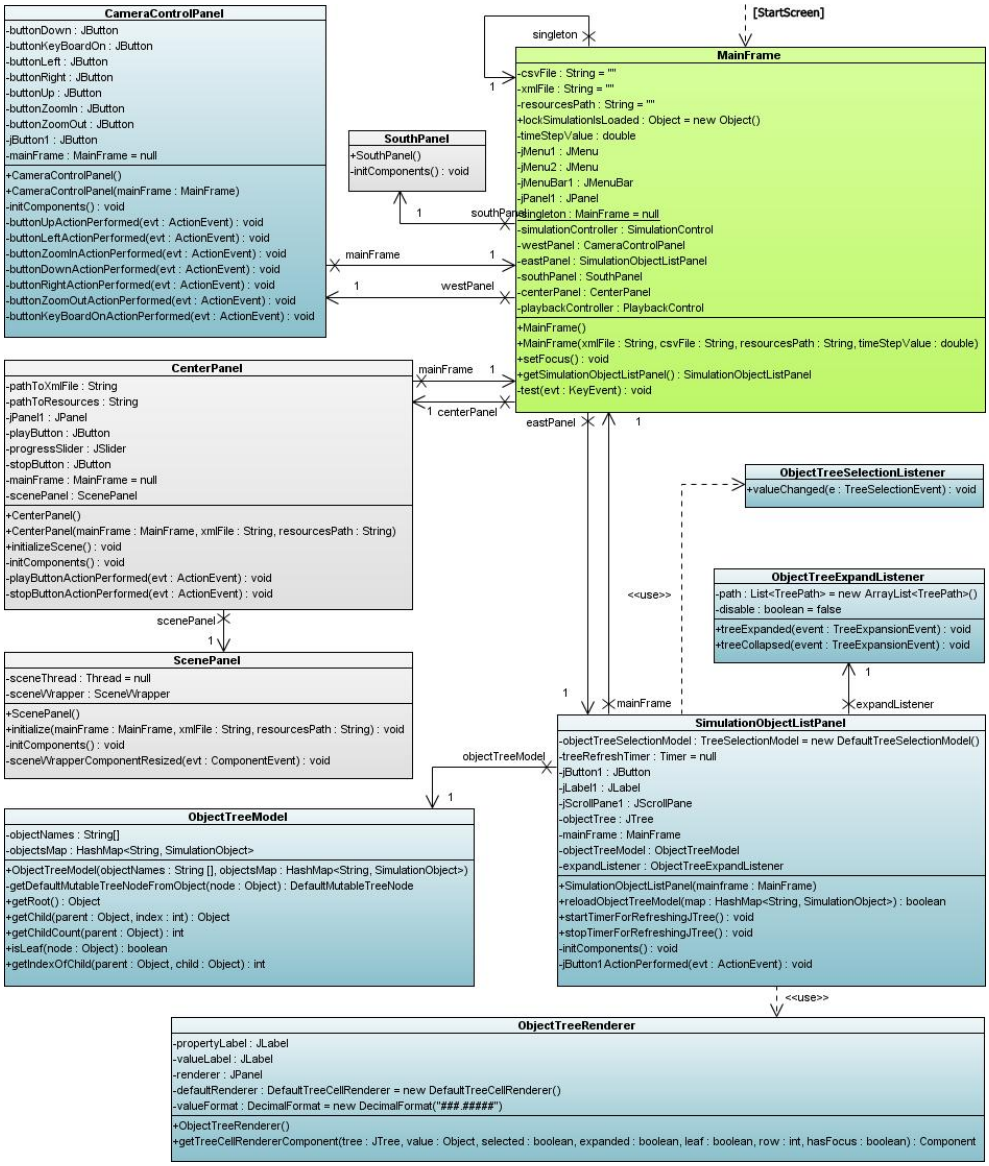


Figure 6.6: Sprint 2 class diagram for simviewer.gui part 2

6.4.1.6. Class descriptions

This section contains tables of new classes and their description. For description of the classes existing prior to this sprint, see 5.4.1.8.

6.4.1.6.a simviewer.data

Class name	Description
NameValue	Holds simple name value pair for use in the simulation object tree
Vector	Holds vector values for use in the simulation object tree

6.4.1.6.b simviewer.gui

Class name	Description
PlaybackControl	The class responsible for updating the state of the simulation controller at the correct rate.
CameraControlPanel	The GUI panel containing camera controls.
SimulationObjectList-Panel	The GUI panel containing the list of objects and values.
ObjectTreeModel	The model for the tree displaying the simulation objects with properties.
ObjectTreeExpand-Listener	The listener used to keep track of which nodes in the tree is expanded, to prevent collapsing when updating values.
ObjectTreeSelection-Listener	The listener used to keep track of which nodes are currently selected and in which order, for later use in analysis/charting.
ObjectTreeRenderer	The renderer used to achieve a custom look with name/value-pairs instead of single values.

6.4.1.6.c simviewer.visualize

Class name	Description
SceneWrapper	Replaces the Swing/AWT part of the previous SceneWindow-class.
SWTSceneWrapper	Replaces the SWT part of the previous SceneWindow-class.
Visualization	Replaces the fhVis/OGRE part of the previous SceneWindow-class.

Table 6.2: Sprint 2 class descriptions

6.5. Development

6.5.1. Making SimViewer Vista compatible

6.5.1.1. Task definition

Although this wasn't a requirement from the customer, the majority of the group's development environment was Microsoft Windows Vista and the problem at hand was interconnected with the whole application, which made it necessary from a development point of view to support it.

The problem discovered in sprint 1 (see 5.5.1.3) was that two of the functions exported from fhVis – getStateObjects() and getStateTags() – would cause a failure when called in Windows Vista. This had to be resolved.

6.5.1.2. Solution

Through investigation the cause of the failure was located. The access violation would arise at every call of the JNI function NewStringUTF().

The first attempt of a solution was to replace this with a call to the function NewString(), as we believed there was an UTF issue in JNI. This did not solve the issue.

Realizing that the ability to create new strings through JNI in Vista was unavailable, we chose to try a different solution which involved creating a basic character array (jchar array), which would then be filled with the strings separated by newlines ('\n').

The creation of character arrays was not crippled by Vista, and the new solution worked as it should, which made the two functions at hand Vista-compatible.

6.5.1.3. Encountered problems

While searching for the cause of the access violation, we had to debug the SceneWindow class which used several threads in a disorderly fashion. Due to this we redesigned this part into separate classes

handling various layers of the visualization window, resulting in the SceneWrapper, SWTSceneWrapper and Visualization classes.

The result was a more manageable and debuggable code structure, which would also help during development at later stages.

6.5.2. *Passing state data to fhVis*

6.5.2.1. Task definition

The application should move from doing live simulation to using the data stored in the CSV-file loaded by the simulation parsing framework.

6.5.2.2. Solution

FhVis has predefined functions for accepting state information instead of doing live simulation. The function we use are simply called render(), which takes the current simulation time and state values as input.

The state values are retrieved from the parsing framework in the Visualization class, which passes these values at a theoretical rate of 25 frames per second (fps). The actual fps may vary due to heavy load, but will never exceed 25.

A frame rate of 25 fps ensures a reasonably smooth visualization, even when large changes occur such as fast moving objects or quick rotation/movement of camera.

The smoothness of the actual simulation is dependent on the time step value specified in the simulation parsing framework, which is user specified at load time. A smaller time step value leads to smoother simulation.

6.5.2.3. Encountered problems

Since we use fhVis through JNI, we anticipated some problems passing values. However, the procedure worked as required, and no other significant problems occurred during development of this task.

6.5.3. *Development of simulation object tree*

This section describes the development process of the simulation object tree. The simulation object tree changes previous implementation of displaying objects and their properties. Simulation objects were represented as items in a list and properties of the selected object were displayed in a separate table.

We decided to change this solution according to the discussion with the customer. Their main requirement was to put all values in a more intuitive structure described below.

6.5.3.1. Task definition

The main purpose of the simulation object tree was displaying object information in a more intuitive way for the user. It needed to cooperate with the simulation output parser and update values during the simulation playback. If possible, the tree structure should stay unchanged when updating values. This meant that we needed to keep nodes expanded after the tree was refreshed and only change leaf values.

Functionality to select one or more values and provide background for calculating functions of values was also required. The actual implementation of calculations would be covered in a later sprint.

6.5.3.2. Solution

According to the task definition we needed to use a tree structure for representing objects and provide collapsing functionality for vector values. There were two issues; preparing structure for the tree to be able to show collapsed and expanded vectors, and provide functionality to select objects from the list and recognize these selections.

6.5.3.2.a Structure for collapsing vectors

We decided to create a helpful structure which contains information about vectors and their values, to minimize the problem of collapsing vectors.

We added helpful arrays with vectors and simple values to the SimulationObject class. We also created classes representing vectors (Vector) and simple values (NameValue).The Vector class contains the following attributes: name, id and an array of simple values. NameValue has two attributes; name and value. For better understanding see the class diagram in section 6.4.1.2. This provides a good structure for creating a model for JTree (described in 6.5.3.2.b).

We added a new method to SimulationObjectParser, collapseVectors(). This method is called every time the parser reads a column of the CSV-file and checks if it belongs to:

- the same vector as the previous column – add this as a simple value to the array of simple values in the vector
- new vector – creates new vector and adds the first value into a simple values array
- simple value – creates a new simple value and adds it to the SimulationObject simple values array

This method is called from the initialize()-method of SimulationObject-Parser as well as from the updateBuffer()-method.

6.5.3.2.b Displaying values in the tree

We are using the Java Swing GUI framework which contains a component called JTree. This component provides main functionality for displaying values in a tree structure. In order to use this component one have to implement classes providing functionality for displaying values in the tree and also functions for intended behavior of the tree (collapsing, refreshing, selecting values). These classes divide the task into sub-tasks listed below:

1. Creating a model for the simulation object tree – this model has to provide functionality for displaying objects and their values in the JTree. It needs to hold the actual values of the simulation objects and properties and it should be able to efficiently update these values during playback (because the values of the objects and its properties may change for each time instant).

2. Tree listeners – the listeners help to remember the state of the tree. It provides functionality to remember which nodes are collapsed which is helpful when updating the nodes. Because we only want to update the nodes which are collapsed (to save system resources).
3. Selection model – Keeps selections of objects in memory. We need to know which objects that are selected in order to be able to generate charts. This part will be postponed to a later sprint.

Creating model for simulation object tree

We used ObjectTreeModel for representing the JTree model. It extends DefaultTreeModel which allows use of the predefined reload()-method. Additionally we need to override the methods listed below:

- getRoot() – returns root of tree
- getChild(Object parent, int index) – returns children number index of parent.
- getChildCount(Object parent) – returns the number of children for parent.
- isLeaf(Object node) – checks If node is containing a simple value. If the node contains a simple value it returns true or false in the opposite case.
- getIndexOfChild(Object parent, Object child) – returns the index of the child, in case the child is not found -1 is returned.

Tree listeners

The application uses ObjectTreeExpand which implements TreeExpandListener. This listener holds the structure of the expanded tree nodes and detects when the user expands or collapses a tree node. When the tree needs to be refreshed the application invokes reloadObjectTreeModel()-method. This method gets the current states of the expanded nodes from TreeExpandListener. It iterates over all expanded nodes and invokes reload.

Selection model

The selection model was implemented only as a skeleton for the next sprint. In the current sprint ObjectTreeSelectionModel provides a method for displaying objects that have been selected when the user clicks on the button “test selection”. This was implemented to prepare for the next sprint issue called “selecting objects from object tree”.

6.5.3.3. Encountered problems

The development of the collapsing vector feature was straight forward. But during the presentation and discussion with the customer we recognized one issue: The collapsing vector feature is choosing candidates for vectors only by looking for an underscore in the name of the property. After discussion with the customer we discovered that there are some properties with underscores that are not vectors. The customer was not sure whether this was an error in the input file or not, but promised to notify us about this as soon as possible.

The biggest challenge appeared during the object tree model development. First version of this model was a class which implemented the TreeModel interface. We realized that the TreeModel interface was not good enough for our purpose; there is no possibility to easily reload the whole model. So we needed another solution to reload the method by ourselves efficiently. We discovered that the best way to do this is by extending the DefaultTreeModel, which provides some functionality for reloading values in a tree. The new version of the object tree was implemented. The most significant change was using a DefaultMutableTreeNode for every node and connecting every node with its children and every child with its parent. This allows the new model to use the reload() function from its super-class.

6.5.3.4. Conclusion

We created a new intuitive user interface for displaying values of objects. This new interface allows a user to see vector values collapsed together. It also allows multiple selections of values, so it will provide

extended functionality for charting. The customer confirmed that this was the desired approach.

On the other hand there are some issues which need to be finished or improved in a later sprint. We need to finish the SelectionModel implementation which will provide the necessary methods for supporting charting functionality.

For improvement we can change visual appearance of the object tree, and make different icons for objects, vectors and values. Now we used a default folder structure. We should test the performance with bigger input files in order to check if the response time is reasonable.

6.5.4. Simulation controller framework – Parser changes

In sprint 2 we got a new example file from SINTEF. This file turned out to be quite different than the previous one regarding timing. First of all, not all times are given in integer values; resulting in our code not working as intended (we assumed integer values). Second of all, the time differences between consecutive frames are not consistent. This was the biggest problem, and lead to many changes throughout the parser.

In order to get consistent time differences between consecutive frames, we let the user choose which time step value he wants in the start dialog. After the user has specified this time step value, a temporary file with all frames in the simulation for the given time step is created. We solved this by letting the program read through the original file, searching for all frames that equals (previous frame + time step) for each frame. For all frames that match directly, we write the lines directly to the temporary file. For all other frames, we have to interpolate our desired values from the frames that are available in the original file. This is done via linear interpolation. This solution was considered best considering that we want maximum performance during playback.

When the temporary file is created, we go on with the buffering as before.

In order to incorporate that time can be double values, we had to do some other modifications to the initialize and update buffer methods. The time variable was changed to double and a separate variable that took care of the number of frames we are processing was added. Additionally we added methods so that we can access both the total simulation time and total number of simulation frames from the simulation control.

When the Java virtual machine (JVM) is shut down, it deletes all the temporary files created during the simulation.

6.5.4.1. Validations of the CSV-file

During program startup the user is requested to specify a path to a valid CSV-file. A file filter makes sure the user is only able to select files that end with .csv.

When the user has specified a valid path to a CSV-file and started the application the file is loaded into memory. While the file is loaded into memory consistency check is done to ensure it's a valid CSV-file, which must follow the following standards:

- The first line contains the object names
- The second line contains a set of attributes for the objects
- The rest of the lines contain a number of floats equal to the number of objects specified in first line.

If the file does not comply with this standard an error dialog is produced and shown to the user. When user confirms the error message the application closes, since we are not able to continue unless the data file is valid.

6.5.4.2. Activity diagram for the parser

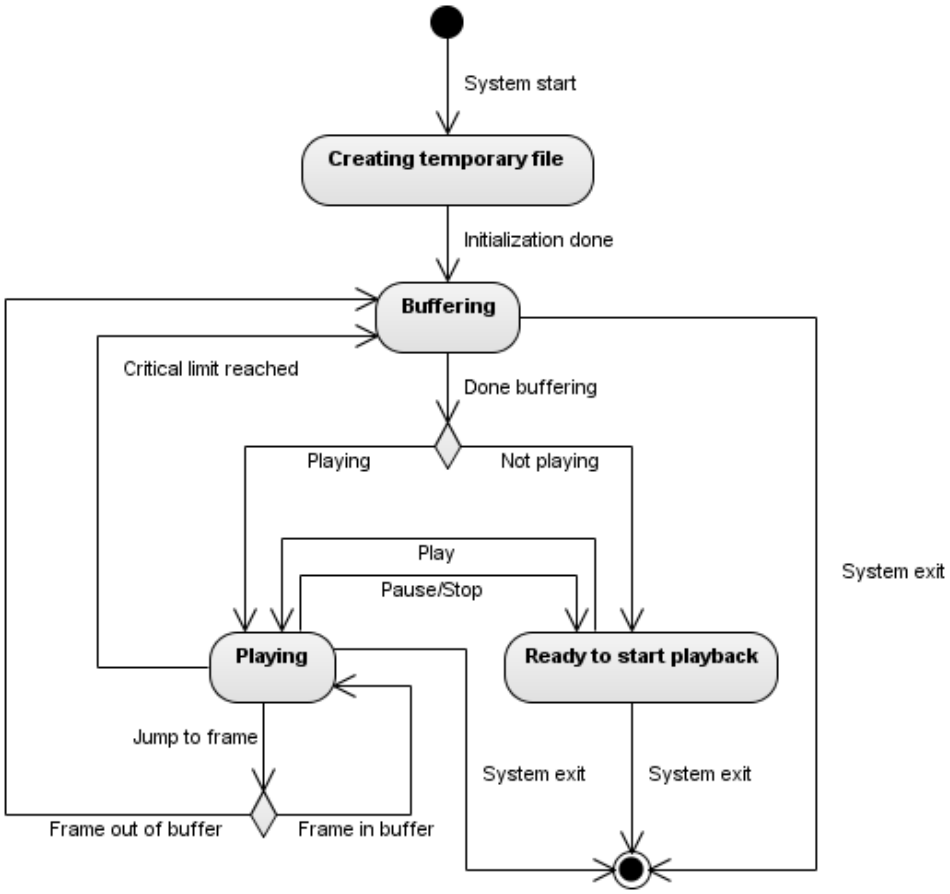


Figure 6.7: Activity diagram for parser

6.5.4.3. Comments to the activity diagram

The activity diagram above shows how the parser works. When the application starts, a temporary file is created. When the temporary file successfully has been created, the program automatically starts buffering to be ready for playback. When buffering is finished the program enters a state where it is ready to start playback.

When the user press the play button playback starts immediately. The user may whenever he/she wants cancel playback by pressing stop. This returns the program in ready to start playback state.

While the simulation is playing the user may jump to a specific frame by pulling the slider. This may result in a state where the current frame is outside the buffered region. This results in the program entering a buffering state. When buffering is finished it will automatically start playing again.

6.6. Testing

ID/Title	1.0 Test of playback simulation (communication with fhVis)
Description	In this test we want to check whether SimViewer is able to send data from the simulation file (CSV file) to fhVis. In order for this to work there need to be functionality for sending the simulation data from the SimViewer to fhVis. During this test we will only use one CSV file that we have received from the customer. The simulation name is "trawltest". The CSV file size is 500 MB and the simulation contains 9 objects and a lot of properties for each object, it should be sufficient to validate whether or not the communication with fhVis works. The simulation should be loaded with help from the startup dialog and when the loading of the CSV file is done the play button should be pressed.
Pre-condition	SimViewer is started up with correct XML and CSV file and the path to the resource folder, buffering is finished and playback is started.
Post-condition	The simulation should play until the end and there should not be any problems while playing. It should be smooth and there should be a feeling of continuity in the simulation.
Comments	It works fine, no lagging.
Pass	Ok

ID/Title	1.1 Test of playback simulation control GUI – Skipping frames
Description	In this test an arbitrary valid simulation should be loaded and one should try to skip frames by moving the slider forwards and backwards. When the slider is moved outside the buffered region, re-buffering should not take more than two seconds. After buffering is done the playback should immediately continue.
Pre-condition	SimViewer is running and simulation playback is running.
Post-condition	The simulation should maximum stop for 2 seconds to re-buffer after a new current frame is specified (slider moved). After this time the simulation should continue from the new spot.
Comments	It works to move forward, but not backward.
Pass	Failed

ID/Title	1.2 Test of playback simulation control GUI – Stop button
Description	In this test a simulation should be loaded and one should try to push the stop button during playback.
Pre-condition	SimViewer is running and simulation playback is running.
Post-condition	The simulation should stop and the slider should move to the start of the playback (start position).
Comments	
Pass	Ok

ID/Title	1.3 Test of playback simulation control GUI – Pause button
Description	In this test a simulation should be loaded and one should try to push the pause button during playback.
Pre-condition	SimViewer is running and simulation playback is running.
Post-condition	The simulation stops at the current frame
Comments	
Pass	Ok

ID/Title	2.0 Test of CSV file validation
Description	Want to check whether SimViewer is able to detect if the specified CSV file is valid or not. Will try with two faulty CSV files: One CSV file that does not contain any data One CSV file that has not enough values according to how many objects there is in the header SimViewer should be started, the dialog where one can input CSV file path should be open. In this dialog the path to one (at a time) faulty CSV file should be specified. The start visualization button should then be pushed.
Pre-condition	SimViewer is running and the startup dialog is open.
Post-condition	SimViewer should display an error message to the user, and when the user confirms this error message the program should be terminated.
Comments	
Pass	Ok

ID/Title	3.0 Test of new simulation output parser
Description	Want to test the new simulation output parser. It had to be redesigned due to some simplification assumed by us. The old simulation parser could only handle integer time indices, but now it should be able to handle float time values. Should load SimViewer with a valid CSV file containing float values as time indices.
Pre-condition	SimViewer is started with a valid CSV file that contains float values as time indices.
Post-condition	There should be no errors during parsing that means after startup processes has finished SimViewer should be ready to start playback.
Comments	
Pass	Ok

ID/Title	4.0 Object list
Description	Want to check if the objects specified in the CSV file occur in the object list. This is done by starting SimViewer with a valid simulation, and then cross checking the object list values with the objects in the CSV file.
Pre-condition	SimViewer is running with a valid simulation loaded.
Post-condition	All objects occurring in object list should be specified in the CSV file. No doublets should occur.
Comments	
Pass	Ok

ID/Title	5.0 Selecting objects from list
Description	Want to test whether it is possible to select simulation objects from object list in the main window of SimViewer. So far no functionalities other than detection of selected objects have been implemented. The only way to check if it works is to check the program output in the console. Should try to select one object first and validate that system detects this by checking the output. Then one should select another object to check if SimViewer is able to detect this.
Pre-condition	SimViewer is running with a valid simulation loaded.
Post-condition	Should print out that a new object has been selected.
Comments	
Pass	Ok

ID/Title	6.0 Display information of simulation objects
Description	Want to test if the object list is able to display correct values for the objects. The values of the objects should change as the simulation is playing. Because object values are changing for each time indices the respective CSV file containing the objects and values for each time instant has to be crosschecked with the values displayed in the SimViewers object list. By inspection we want to validate that for some arbitrary time instants and objects values these are according to the values specified in the CSV file.
Pre-condition	SimViewer is running and the simulation is running.
Post-condition	Have to pause the playback to inspect object values and arbitrary time instants. The values have to match.
Comments	Seems to work, it's not easy to check for all object values and time instants because it is incredibly many values.
Pass	Ok

6.7. Conclusion

The main task in this sprint was to interpolate an output file from fhSim and show visualization playback instead of simulation. We managed to pass information about objects through NativeBridge to fhVis. After that fhVis initializes a scene with all objects.

We put information about objects into the tree so the customer can collapse/expand nodes with the information he wants. The tree structure allows us to implement updating only of nodes that are expanded, which saves a lot of computing time.

For future charting we prepared support for selecting objects/attributes. During this sprint we met a lot of problems that appear only in Vista with JNI but all of them were solved.

Chapter 7
Sprint 3

7.1. Chapter introduction

This chapter describes the processes that took place during sprint 3.

The purpose of this chapter is to document the progress, results and acquired knowledge during sprint 3 of the project.

What is covered here took place in the time period between October 20th and October 31st, 2008.

This chapter has the following contents:

- Sprint backlog – Includes the elements where the focus of this sprint was placed.
- Design – Any system design that took place during this sprint.
- Development – Describes the development that took place during this sprint.
- Testing – Describes a set of tests used to make sure that the application functions according to the requirements
- Conclusion – A summary of the progress and problems encountered during this sprint.

7.2. Sprint backlog

Item #	Description (with reference to req. number)	Hours est.	Risk est.	Respon-sible	Assigned members
8	2.2 Time stepping	50	M	Rolf	Rolf
12	3.7 Display time plot of force of selected object				
12.1	- Charting	30	M	Pavel	Jiri and Pavel

12.2	- Object list and GUI	25	M	Jiri	Jiri and Pavel
12.3	- Parser changes	25	M	Rolf	Rolf and Erik
13	4.3 Save charts to image file	5	L	Pavel	Pavel
14	2.4 Following camera	30	M	Jørgen	Jørgen
16	4.1 Take screenshot during the simulation	15	L	Jørgen	Jørgen
	Total – sprint 3	**180**			

Table 7.1: Sprint 3 backlog

7.3. Sprint Gantt chart

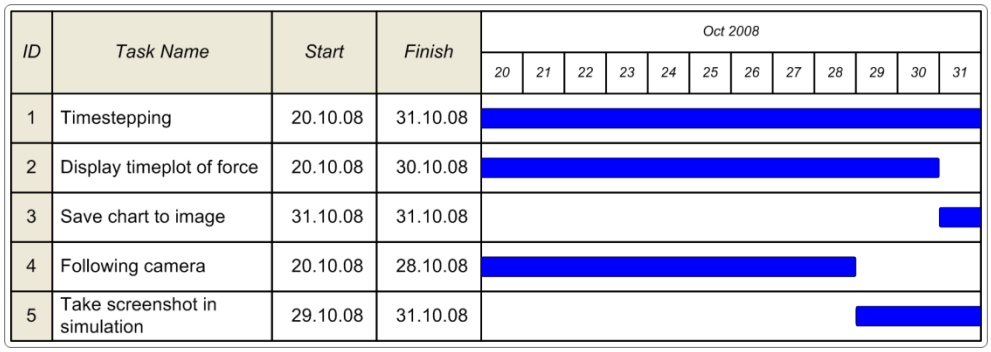


Figure 7.1: Sprint 3 Gantt chart

7.4. Design

7.4.1. Package and class diagrams

Following are the package and class diagrams for sprint 3. Some packages have not undergone any changes in this sprint, and are not

included here. See sections 5.4.1 and 6.4.1 for diagrams of the previous sprints.

New classes and packages are marked with blue.

Significantly altered classes are marked with green.

7.4.1.1. Package diagram

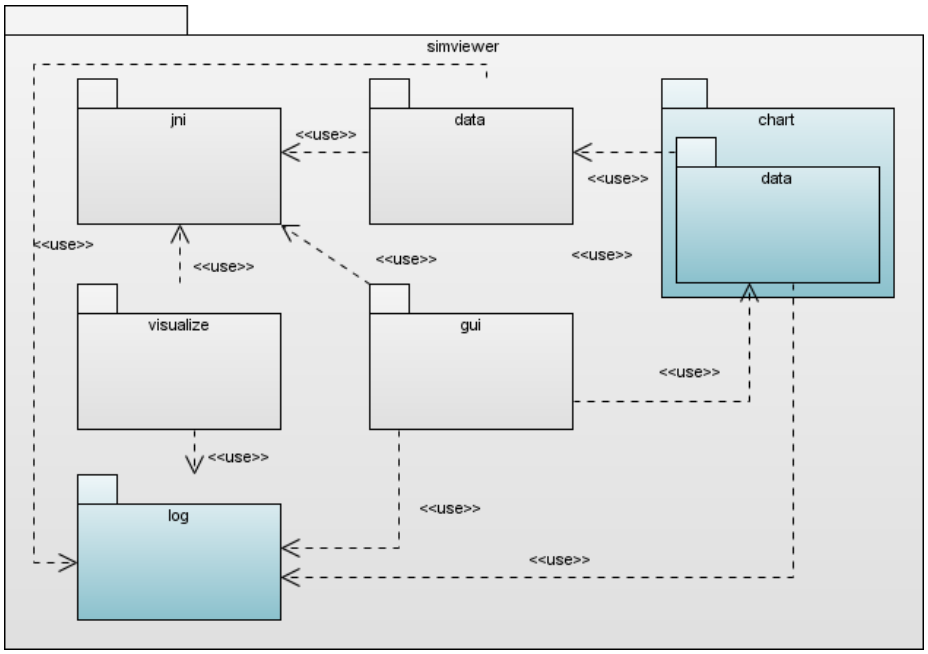


Figure 7.2: Sprint 3 package diagram

7.4.1.2. Class diagram for simviewer.jni

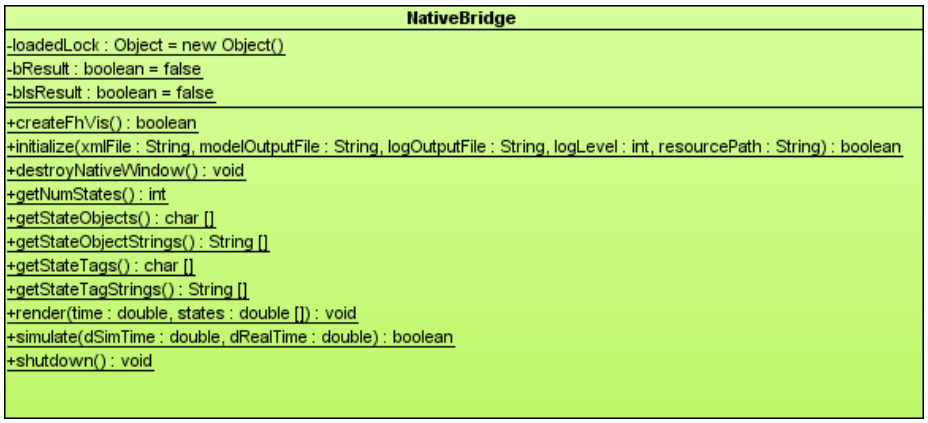

NativeBridge
-loadedLock : Object = new Object()
-bResult : boolean = false
-blsResult : boolean = false
+createFhVis() : boolean
+initialize(xmlFile : String, modelOutputFile : String, logOutputFile : String, logLevel : int, resourcePath : String) : boolean
+destroyNativeWindow() : void
+getNumStates() : int
+getStateObjects() : char []
+getStateObjectStrings() : String []
+getStateTags() : char []
+getStateTagStrings() : String []
+render(time : double, states : double []) : void
+simulate(dSimTime : double, dRealTime : double) : boolean
+shutdown() : void

Figure 7.3: Sprint 3 class diagram for simviewer.jni

7.4.1.3. Class diagram for simviewer.data

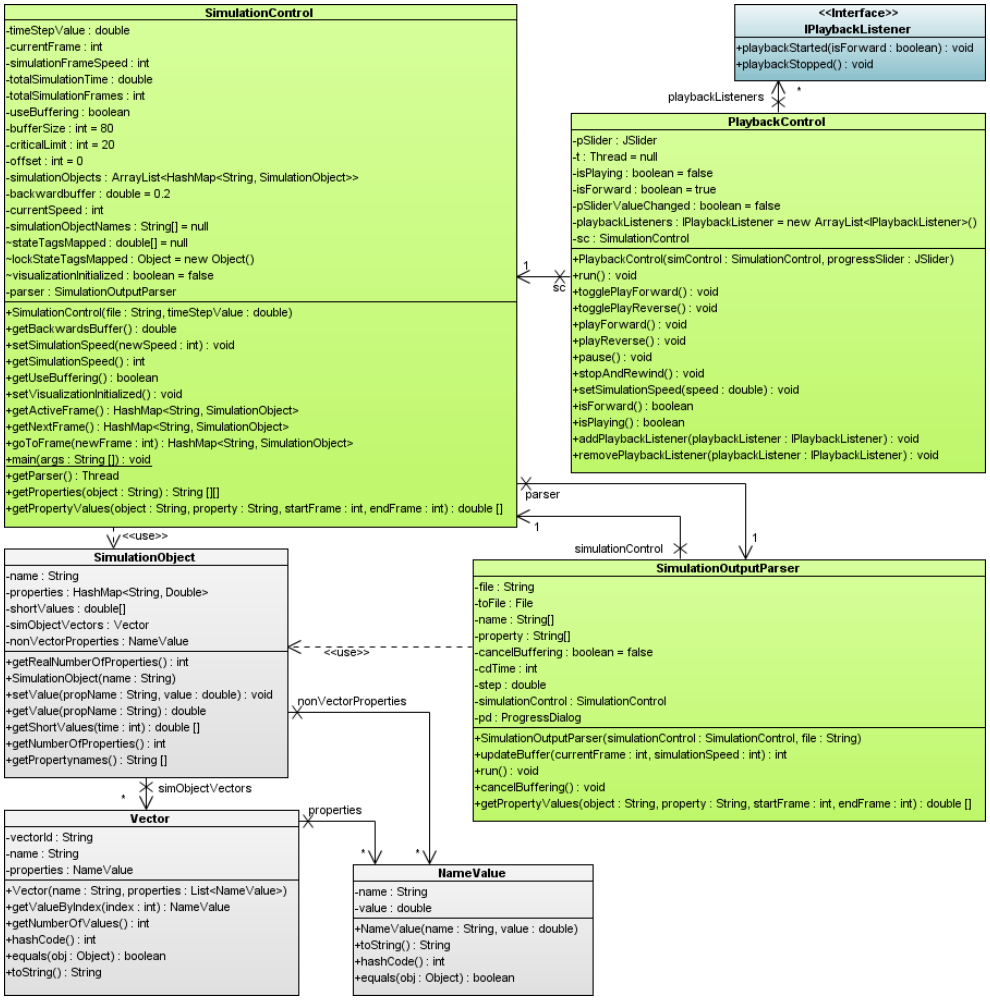


Figure 7.4: Sprint 3 class diagram for simviewer.data

7.4.1.4. Class diagram for simviewer.visualize

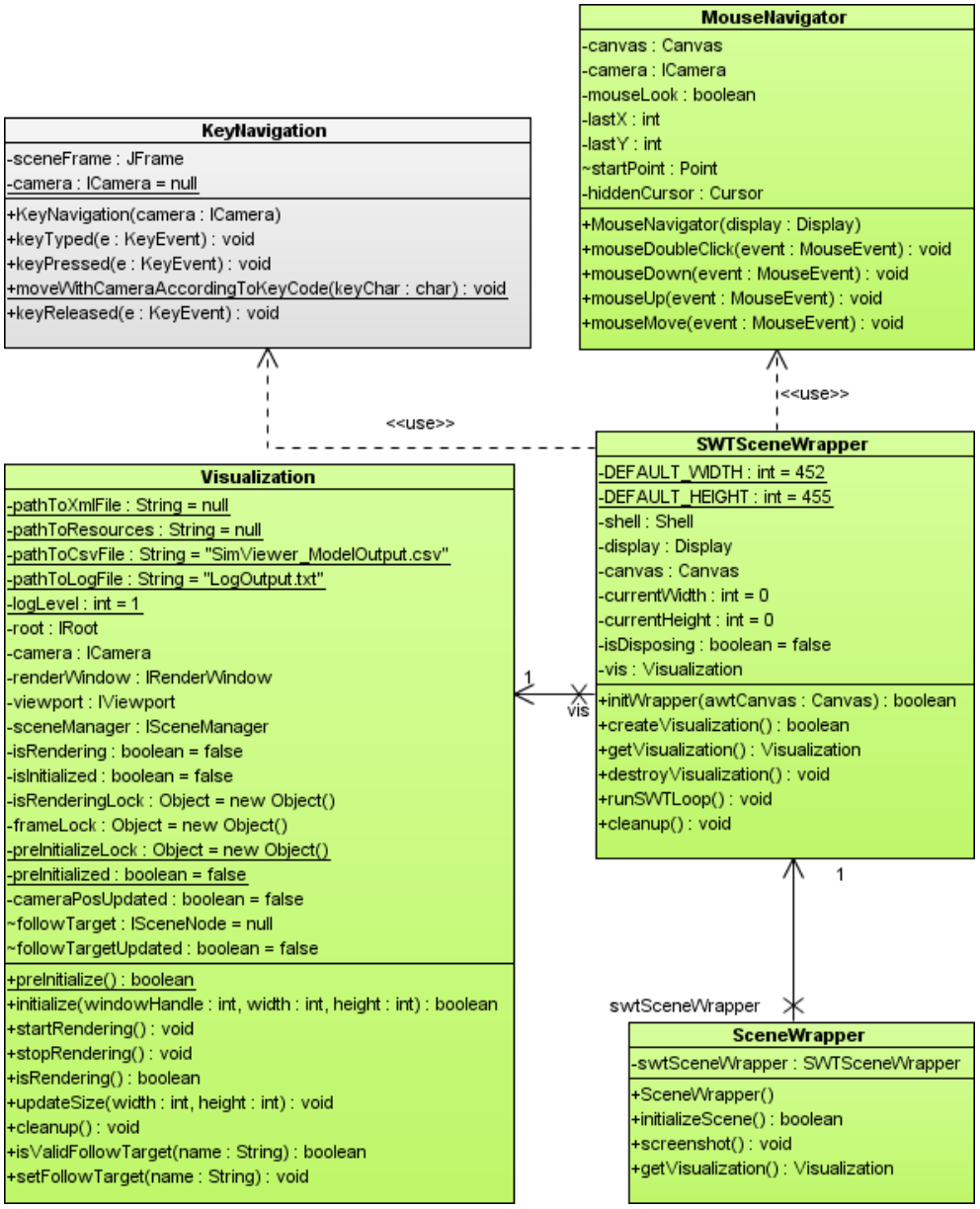


Figure 7.5: Sprint 3 class diagram for simviewer.visualize

7.4.1.5. Class diagram for simviewer.gui part 1

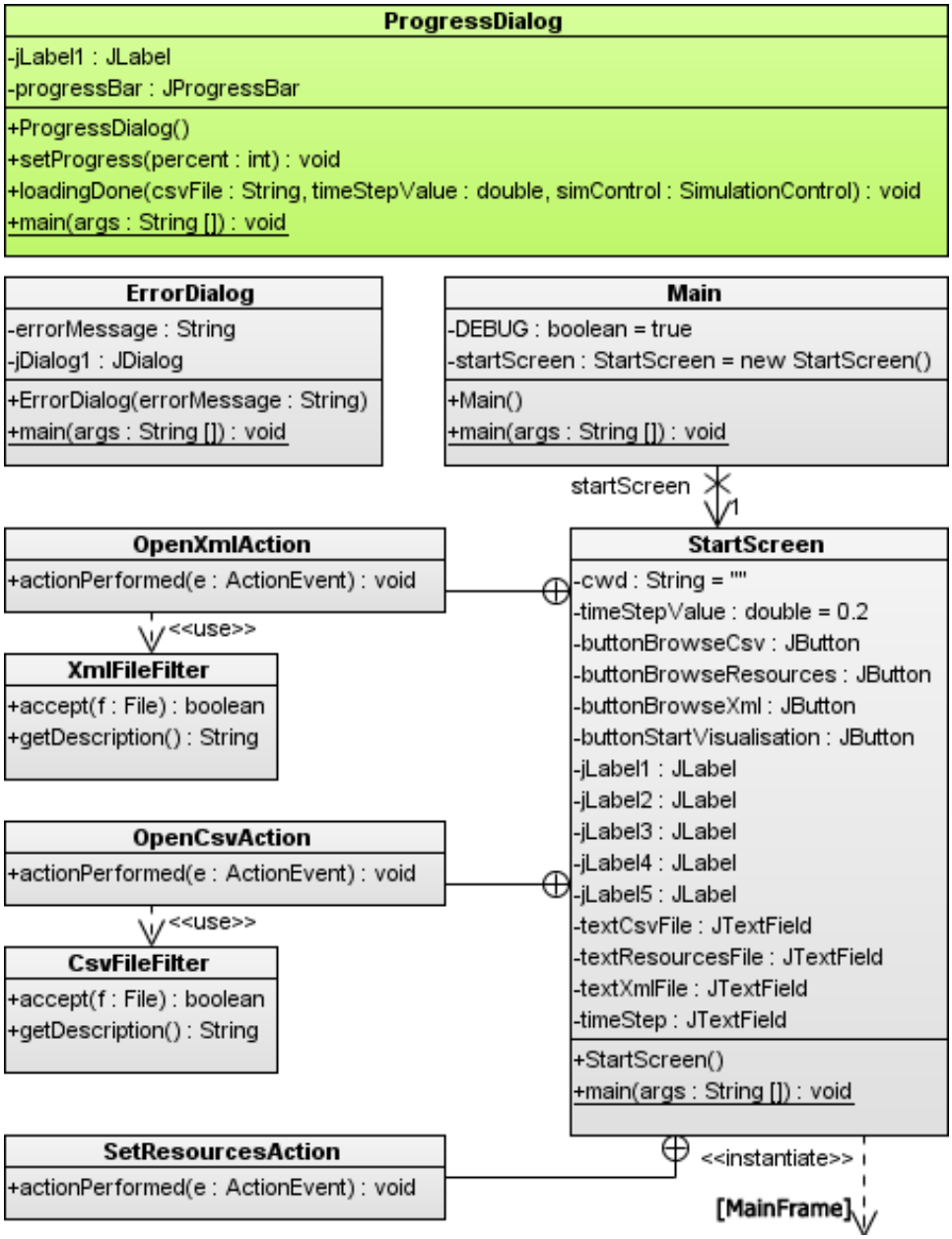


Figure 7.6: Sprint 3 class diagram for simviewer.gui part 1

7.4.1.6. Class diagram for simviewer.gui part 2

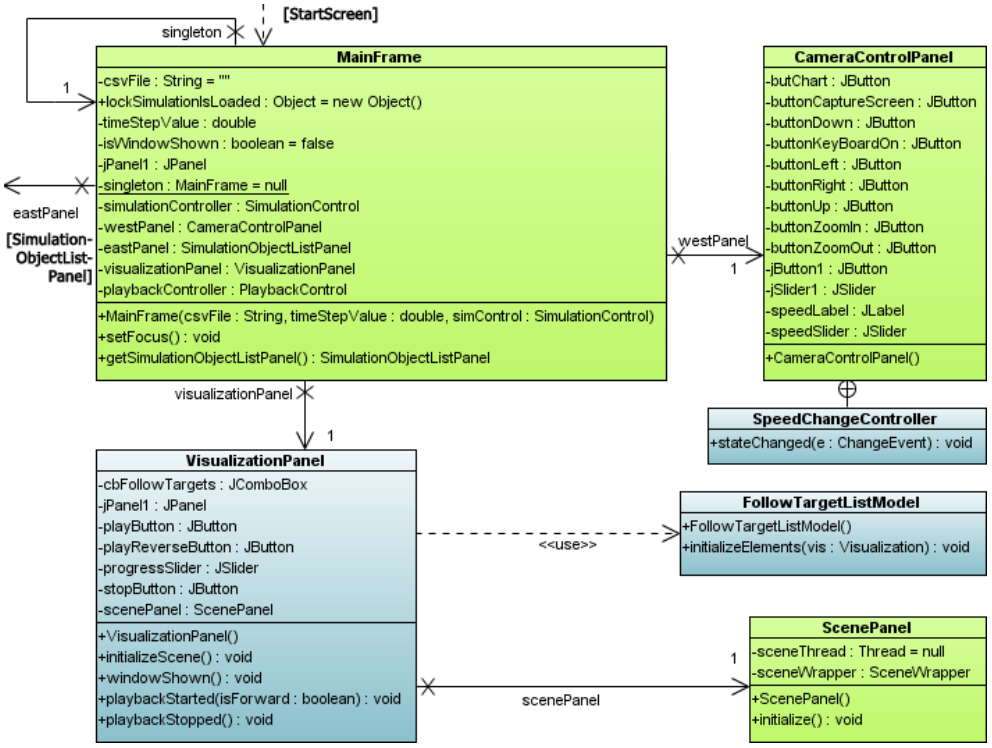


Figure 7.7: Sprint 3 class diagram for simviewer.gui part 2

7.4.1.7. Class diagram for simviewer.gui part 3

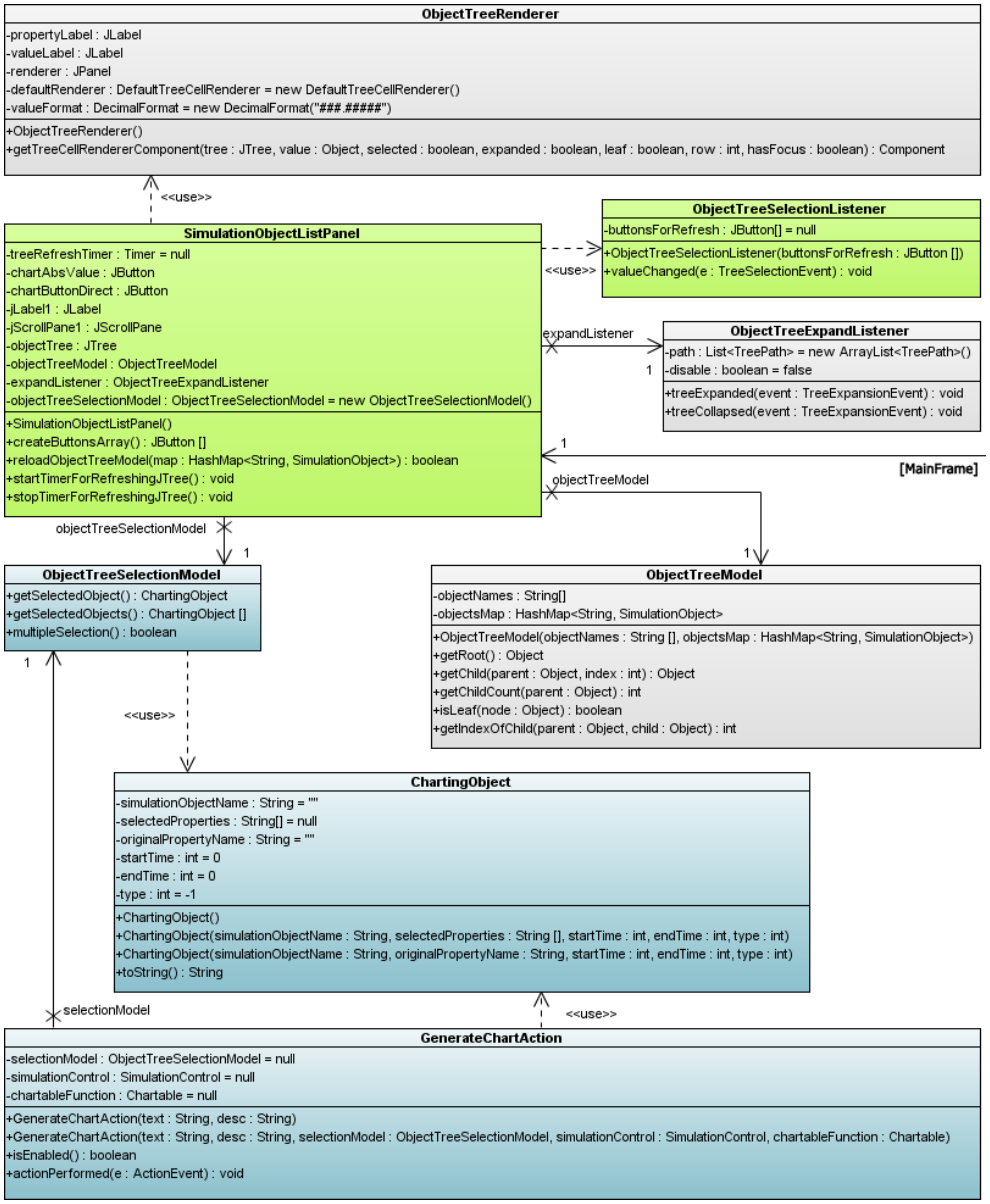

ObjectTreeRenderer
-propertyLabel : JLabel
-valueLabel : JLabel
-renderer : JPanel
-defaultRenderer : DefaultTreeCellRenderer = new DefaultTreeCellRenderer()
-valueFormat : DecimalFormat = new DecimalFormat("###.#####")
+ObjectTreeRenderer()
+getTreeCellRendererComponent(tree : JTree, value : Object, selected : boolean, expanded : boolean, leaf : boolean, row : int, hasFocus : boolean) : Component
<<use>>
SimulationObjectListPanel
-treeRefreshTimer : Timer = null
-chartAbsValue : JButton
-chartButtonDirect : JButton
-jLabel1 : JLabel
-jScrollPane1 : JScrollPane
-objectTree : JTree
-objectTreeModel : ObjectTreeModel
-expandListener : ObjectTreeExpandListener
-objectTreeSelectionModel : ObjectTreeSelectionModel = new ObjectTreeSelectionModel()
+SimulationObjectListPanel()
+createButtonsArray() : JButton []
+reloadObjectTreeModel(map : HashMap<String, SimulationObject>) : boolean
+startTimerForRefreshingJTree() : void
+stopTimerForRefreshingJTree() : void
ObjectTreeSelectionListener
-buttonsForRefresh : JButton[] = null
+ObjectTreeSelectionListener(buttonsForRefresh : JButton [])
+valueChanged(e : TreeSelectionEvent) : void
<<use>>
expandListener
ObjectTreeExpandListener
-path : List<TreePath> = new ArrayList<TreePath>()
-disable : boolean = false
+treeExpanded(event : TreeExpansionEvent) : void
+treeCollapsed(event : TreeExpansionEvent) : void
1
[MainFrame]
objectTreeModel
objectTreeSelectionModel
ObjectTreeSelectionModel
+getSelectedObject() : ChartingObject
+getSelectedObjects() : ChartingObject []
+multipleSelection() : boolean
ObjectTreeModel
-objectNames : String[]
-objectsMap : HashMap<String, SimulationObject>
+ObjectTreeModel(objectNames : String [], objectsMap : HashMap<String, SimulationObject>)
+getRoot() : Object
+getChild(parent : Object, index : int) : Object
+getChildCount(parent : Object) : int
+isLeaf(node : Object) : boolean
+getIndexOfChild(parent : Object, child : Object) : int
ChartingObject
-simulationObjectName : String = ""
-selectedProperties : String[] = null
-originalPropertyName : String = ""
-startTime : int = 0
-endTime : int = 0
-type : int = -1
+ChartingObject()
+ChartingObject(simulationObjectName : String, selectedProperties : String [], startTime : int, endTime : int, type : int)
+ChartingObject(simulationObjectName : String, originalPropertyName : String, startTime : int, endTime : int, type : int)
+toString() : String
selectionModel
GenerateChartAction
-selectionModel : ObjectTreeSelectionModel = null
-simulationControl : SimulationControl = null
-chartableFunction : Chartable = null
+GenerateChartAction(text : String, desc : String)
+GenerateChartAction(text : String, desc : String, selectionModel : ObjectTreeSelectionModel, simulationControl : SimulationControl, chartableFunction : Chartable)
+isEnabled() : boolean
+actionPerformed(e : ActionEvent) : void

Figure 7.8: Sprint 3 class diagram for simviewer.gui part 3

7.4.1.8. Class diagram for simviewer.chart.data

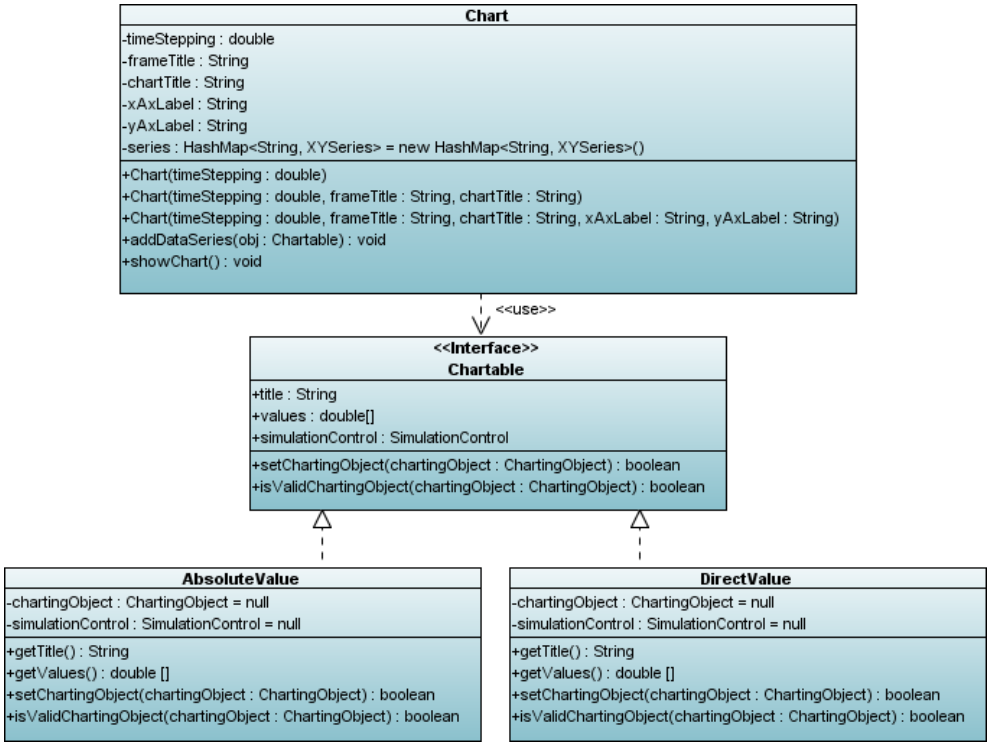


Chart
-timeStepping : double
-frameTitle : String
-chartTitle : String
-xAxLabel : String
-yAxLabel : String
-series : HashMap<String, XYSeries> = new HashMap<String, XYSeries>()
+Chart(timeStepping : double)
+Chart(timeStepping : double, frameTitle : String, chartTitle : String)
+Chart(timeStepping : double, frameTitle : String, chartTitle : String, xAxLabel : String, yAxLabel : String)
+addDataSeries(obj : Chartable) : void
+showChart() : void
<<use>>
<<Interface>>
Chartable
+title : String
+values : double[]
+simulationControl : SimulationControl
+setChartingObject(chartingObject : ChartingObject) : boolean
+isValidChartingObject(chartingObject : ChartingObject) : boolean
AbsoluteValue
-chartingObject : ChartingObject = null
-simulationControl : SimulationControl = null
+getTitle() : String
+getValues() : double []
+setChartingObject(chartingObject : ChartingObject) : boolean
+isValidChartingObject(chartingObject : ChartingObject) : boolean
DirectValue
-chartingObject : ChartingObject = null
-simulationControl : SimulationControl = null
+getTitle() : String
+getValues() : double []
+setChartingObject(chartingObject : ChartingObject) : boolean
+isValidChartingObject(chartingObject : ChartingObject) : boolean

Figure 7.9: Sprint 3 class diagram for simviewer.chart.data

7.4.1.9. Class diagram for simviewer.log

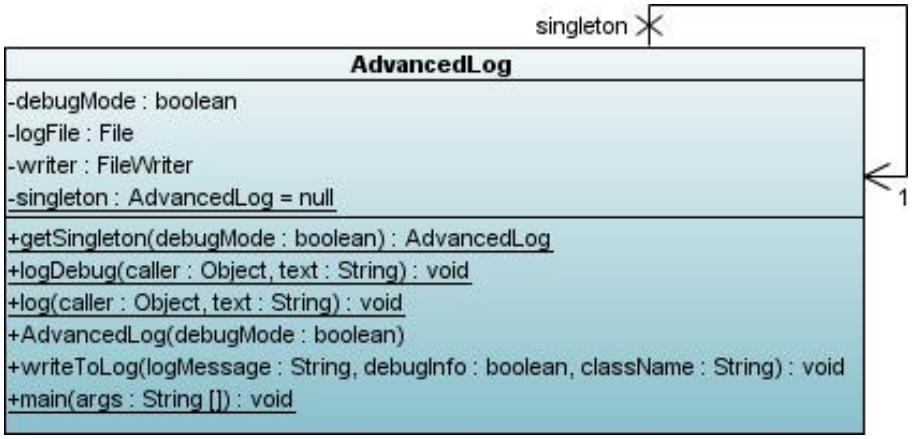

singleton
AdvancedLog
-debugMode : boolean
-logFile : File
-writer : FileWriter
-singleton : AdvancedLog = null
+getSingleton(debugMode : boolean) : AdvancedLog
+logDebug(caller : Object, text : String) : void
+log(caller : Object, text : String) : void
+AdvancedLog(debugMode : boolean)
+writeToLog(logMessage : String, debugInfo : boolean, className : String) : void
+main(args : String []) : void
1

Figure 7.10: Sprint 3 class diagram for simviewer.log

7.4.1.10. Package descriptions

7.4.1.10.a simviewer.chart.data

This package contains the classes for the charting functionality.

7.4.1.10.b simviewer.log

This package contains the class for the log functionality.

7.4.1.11. Class descriptions

7.4.1.11.a simviewer.data

Class name	Description
IPlaybackListener	Interface for classes that need notification of playback events, such as start and stop.

7.4.1.11.b simviewer.gui

Class name	Description
VisualizationPanel	Panel replacing the CenterPanel. Contains all the visualization GUI controls.
FollowTargetListModel	Model for the follow target drop down list. Uses the Visualization class to determine valid follow targets.
SpeedChangeController	Controller for the speed change GUI controls.
ObjectTreeSelectionModel	Selection model for the simulation object tree. Keeps track of user selections.
ChartingObject	Class for passing selection data to charting framework.
GenerateChartAction	Class for generating GUI actions from charting functions.

7.4.1.11.c simviewer.chart.data

Class name	Description
Chart	The class responsible for initializing and passing data to JFreeChart.
Chartable	An interface representing a charting function.

AbsoluteValue	Implementation of Chartable for absolute value of vectors.
DirectValue	Implementation of Chartable for single values.

7.4.1.11.d simviewer.log

Class name	Description
AdvancedLog	Singleton class with useful logging functionality for both errors and debug messages.

Table 7.2: Sprint 3 class descriptions

7.5. Development

7.5.1. Parser

In this sprint we did not make any major design changes to the parser. However, to provide values from the output file to the charting functionality, we had to make a new method in the parser for fetching these. The reason for this is that since not all values are in memory due to the buffering, we have to read the values from file.

We therefore created a method which takes the object name, parameter name and start and end frame as input. This method then validates these values, reads through the file, and makes an array of the selected values. This array is returned as the result.

7.5.2. Adding functionality to the simulation object tree

This section describes changes in the simulation object tree. It contains all the necessary changes for calculating charts from selected values in the tree.

7.5.2.1. Task definition

Create new functionality in the simulation object tree which allows the user to select one or more objects for charting. New functions should provide all necessary data for getting time plot of objects values. This data will be used as a base for reading and calculating values for the final chart. This issue is described in section 7.5.3.

7.5.2.2. Solution

This task is tightly connected with the part that reads data for charting and calculates values for the final chart. So it needs to provide a suitable solution for reading values. This fact was taken into account and for this purpose the class ChartingObject was created. This class holds all necessary information for getting value sets of properties for charting. The class contains these important values:

- startTime, endTime – for the chart to be able to prepare x-axis and for the property reader to be able to find the right start line and end line.
- originalPropertyName – this value holds the real name of the property in the CSV-file
- type, which can be one of the following:
  - object (e.g. "Vessel")
  - vector (e.g. "Vel", representing velocity)
  - nameValue (e.g. "Vel_1", representing a value of the velocity vector)
- simulationObjectName – holds the name of a simulation object which is necessary for reading value because there could be more values with the same name in the CSV-file, but there are no properties with an identical name for one object.
- selectedProperties[] – array of selected properties which is used only for vectors, when the user selects a vector then this holds all components of the vector.

The class described above is only an object passed to the charting framework, but it was necessary to update the classes working with the simulation object tree. The program needed some kind of interface

providing functionality for getting this ChartingObject filled with values according to the selection made by the user.

For this purpose the class ObjectTreeSelectionModel was created. It extends DefaultTreeSelectionModel and contains four functions for getting ChartingObjects according to selection made by the user.

- getChartingObjectForSpecificTreePath() – private method which returns a charting object according to the TreePath passed as a parameter. It parses the specified TreePath and fills the ChartingObject. Additionally, it provides an interface for two other methods.
- getSelectedObject() – public method which returns a single ChartingObject. This method is called when there's a single selection in the tree.
- getSelectedObjects() – public method which returns an array of ChartingObjects. This method is called when there are multiple selections in the tree.
- multipleSelection() – returns true if there are multiple selections.

7.5.2.3. Conclusion

Development of this part was done concurrently with the development of the charting framework, of which both were designed to integrate gracefully with each other.

7.5.3. Charting

7.5.3.1. Task definition

The task was to implement the functionality to create a time plot of selected object value(s) and save it to an image file.

7.5.3.2. Solution

For this functionality we have decided to use the Java open source library JFreeChart. This is released under the Lesser General Public

License (LGPL) which permits use in proprietary applications. This made it possible for us to use it in this project.

Using this library saved us plenty of time instead of writing our own library. It's easy to use and already contains some other functionality we needed, such as saving chart to an image or zooming in a chart.

7.5.3.2.a Display time plot of selected object

Before we started to implement this functionality, we wanted to focus on the design of this package. We knew that it's not going to be used just for time plot of one selected object, but also for displaying time plot of a function among objects such as distance, angle etc.

It is also likely that the customer may need to implement other functions in the future. Therefore we wanted to make the charting framework expandable, and decided to create the new interface Chartable. This makes adding new functions for charting as simple as possible. Also it keeps creating new function separated from the GUI or any controller of charts.

Because of this requirement, new functionality in the simulation controller had to be added for getting all values of a given object from the input file.

7.5.3.2.b Save charts to image file

Because of the built in functionality for saving charts to image files, this requirement was covered by the JFreeChart library.

7.5.4. Following camera

7.5.4.1. Task definition

The camera should be able to follow an object specified by the user.

7.5.4.2. Solution

A drop down list was added to the GUI containing all objects that are possible to follow. The list is filled with the names of the simulation objects, which are checked through the Visualization class to see if there exists a follow node for each simulation object. This is done through the method isValidFollowTarget().

To find a follow node, we make the OGRE scene manager search for a node named *name* + "FollowNode". If this exists, the simulation object is possible to follow. This naming convention was an existing feature of fhVis.

When the user selects an object to follow, the model notifies the Visualization class through the function setFollowTarget(), which then makes sure that the camera coordinates are updated before each frame.

7.5.4.3. Encountered problems

An issue related to the usage of OGRE in Java made it impossible to get a callback prior to the actual render of the scene.

The render()-method of fhVis both updates states, which includes object positions, as well as rendering the scene.

The combination of these two facts resulted in the inability to update the camera coordinates at the right time, which should have been between state update and scene rendering.

This forced us to update the camera coordinates before updating states, which results in the camera always lagging one frame behind. When the scene is static, or only small changes occur, the effect of this is not noticeable. However, during larger changes, either due to high speeds or slow rendering, the effect is considerable.

The solution to this problem required the customer to alter the fhVis library, which was agreed on the sprint review meeting. This issue will then be solved during the next sprint.

7.5.5. Screen shot functionality

7.5.5.1. Task definition

The user should be able to capture the current frame to an image stored on disk.

7.5.5.2. Solution

By using AWT's screen capture functionality (in Robot), and calculating the rectangle where the visualization is displayed, we were able to capture the scene frame.

The user can press a button to activate the screen capture, which captures the current frame, and asks for a storage location.

The image is then saved to the specified location – either as JPEG or PNG – using AWT's imaging functionality.

7.5.5.3. Encountered problems

The planned solution to this task was to use the built in functionality in OGRE to save the current frame to disk. But again, the function for storing buffer failed in an access violation, and was therefore not available.

An alternative solution with rendering the scene to an off-screen texture buffer was tried as well. This would also have enabled the possibility to store renders of arbitrary size (only limited by graphic memory), which had been suggested as a desired extra feature by the customer.

This solution needed to be done on the C++-side of the application, but ended in an access violation as well.

The possible reason for these access violations is the combined usage of OGRE in C++ and Java at the same time, which was not the original intent of the Ogre4j-library.

Since none of the OGRE solutions worked, we decided to use the backup solution of using AWT.

7.5.6. Log functionality

7.5.6.1. Task definition

This package was created to implement a set of tools to be able to trace program states and satisfy the non-functionality requirement about serviceability. It contains only one class which provides methods for writing log messages to a file.

7.5.6.2. Solution

The log creates a file in the temporary directory of the user account on the operating system. This file is not deleted after program exit.

The log keeps track of the class the message was sent from and the time of the log event. Each time program state changes, i.e. when program enters an exception block or when something unexpected happens, one can make sure this event is written to the log so that an engineer or advanced user later can check the log to see what actually happened in this execution of the program.

It includes a possibility to distinguish between normal program executions and debug mode. This means it can filter out all unnecessary debug information when not running in debug mode to save system resources.

7.6. Testing

ID/Title	1.0 Testing of time stepping
Description	In this test we want to test whether the functionality for jumping in time during the simulation works correctly. We shall be able to jump to any position in the simulation as long as it is in pause mode. Every object in the simulation scene shall be in correct state at the new possession.
Pre-condition	Initialized simulation, and in pause mode.
Post-condition	Initialized simulation, and in pause mode, but at a different time in the simulation than before the test.
Comments	
Pass	Ok

ID/Title	2.0 Following camera – Start following
Description	The test should check if the camera in the visualization area follows the correct object when asked for it. We shall be able to pick an object from the list of objects to follow and the camera should follow it.
Pre-condition	Initialized simulation, in play mode and with free camera.
Post-condition	Initialized simulation, in play mode and with the camera following the given object.
Comments	A bit choppy. The customer thinks it is OK. Might get fixed in the last sprint.
Pass	Ok

ID/Title	2.1 Following camera – Stop following
Description	The test should check if the camera in the visualization area will stop following the object when asked for it. We shall be able to pick “free” from the list of objects to follow and the camera should then be in free mode.
Pre-condition	Initialized simulation, in play mode and with the camera following an object.
Post-condition	Initialized simulation, in play mode and with the camera in free mode
Comments	
Pass	Ok

ID/Title	3.0 Take screen shot
Description	In this test we want to check the functionality for taking a screen shot. The application should save the image file at the given path and the picture should be of the frame that was playing when the user presses the screen shot button.
Pre-condition	Initialized simulation.
Post-condition	Initialized simulation, with screen shot saved to correct path.
Comments	
Pass	Ok

ID/Title	4.0 Display chart of direct value
Description	This test should test whether the chart for direct value is displayed in a correct way. We shall be able to select an object, vector or value and make a chart of the direct value during the simulation time. The chart shall be displayed in a separate window.
Pre-condition	Initialized simulation
Post-condition	Initialized simulation, with new chart window.
Comments	The charts seems to be Ok
Pass	Ok

ID/Title	4.1 Display chart of absolute value
Description	This test should test whether the chart for a direct value is displayed in a correct way. We shall be able to select an object, vector or value and make a chart of the absolute value during the simulation time. The chart shall be displayed in a separate window.
Pre-condition	Initialized simulation
Post-condition	Initialized simulation, with new chart window.
Comments	The charts seems to be Ok
Pass	Ok

ID/Title	5.0 Save image of chart
Description	The user should get a prompt for a place to save the image. The application should save the image file at the given path and the picture should be the same as displayed on screen.
Pre-condition	Initialized simulation, with open chart window
Post-condition	Initialized simulation, with chart window and saved image of chart
Comments	
Pass	Ok

7.7. Conclusion

The hardest task of this sprint was time stepping of visualization. We had to make changes in the simulation controller and not just because of time stepping but also because of having an effective way to get all values of the selected object during the whole visualization.

For the Charting functionality we prepared the Chartable interface that should make it easy for the customer to add new functions for charting. During the sprint we encountered some problems calling OGRE functions through NativeBridge. Therefore we had to make capturing screen shots on Java side which has some disadvantages. This issue also created some problems with following camera.

The solution to this problem required the customer to alter the fhVis library. At the end of the sprint we discovered that time stepping wasn't working correctly, so we planned to include it in the next sprint.

Chapter 8
Sprint 4

8.1. Chapter introduction

This chapter describes the processes that took place during sprint 4.

The purpose of this chapter is to document the progress, results and acquired knowledge during sprint 4 of the project.

What is covered here took place in the time period between November 3rd and November 14th, 2008.

Since this was the last sprint, the development had a lower priority than in earlier sprints, and the primary focus was the final presentation and preparing the project report for delivery.

This chapter has the following contents:

- Sprint backlog – Includes the elements where the focus of this sprint was placed.
- Design – Any system design that took place during this sprint.
- Development – Describes the development that took place during this sprint.
- Testing – Describes a set of tests used to make sure that the application functions according to the requirements
- Conclusion – A summary of the progress and problems encountered during this sprint.

8.2. Sprint backlog

Item #	Description (with reference to req. number)	Hours est.	Risk est.	Respon-sible	Assigned members
8	2.2 Time stepping	50	M	Rolf	Rolf
15	3.8 Display time plot of functions of signals available in multiple objects	80	H	Pavel	Jiri and Pavel
	Total – sprint 4	**130**			

Table 8.1: Sprint 4 backlog

8.3. Sprint Gantt diagram

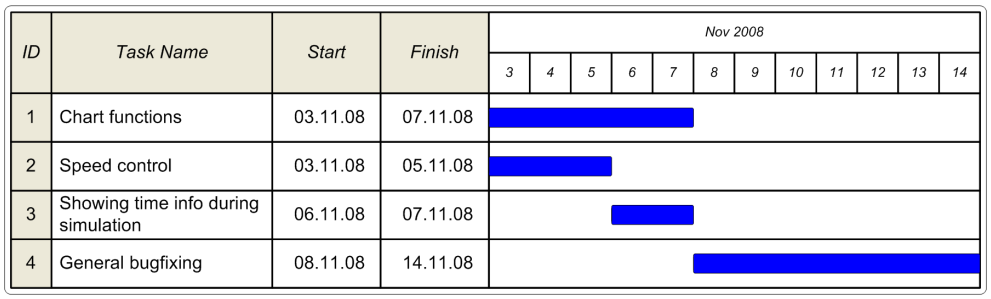


Figure 8.1: Sprint 4 Gantt chart

8.4. Design

8.4.1. Class diagrams

8.4.1.1. Class diagram for simviewer.jni

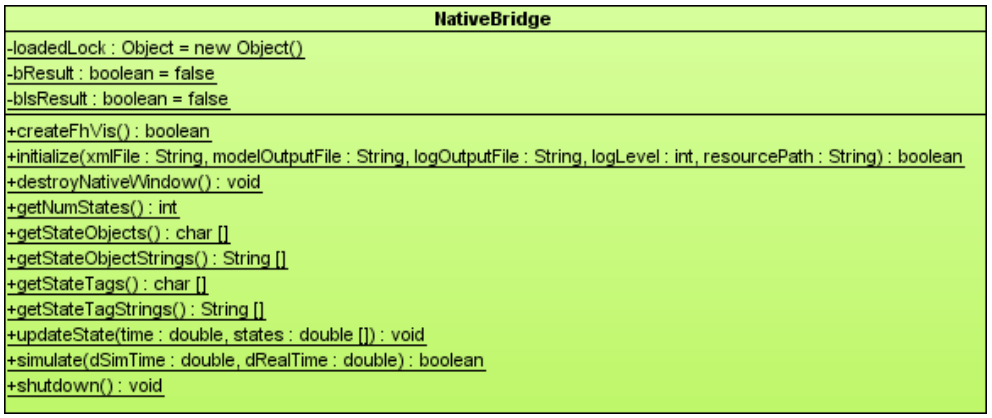

NativeBridge
-loadedLock : Object = new Object()
-bResult : boolean = false
-blsResult : boolean = false
+createFhVis() : boolean
+initialize(xmlFile : String, modelOutputFile : String, logOutputFile : String, logLevel : int, resourcePath : String) : boolean
+destroyNativeWindow() : void
+getNumStates() : int
+getStateObjects() : char []
+getStateObjectStrings() : String []
+getStateTags() : char []
+getStateTagStrings() : String []
+updateState(time : double, states : double []) : void
+simulate(dSimTime : double, dRealTime : double) : boolean
+shutdown() : void

Figure 8.2: Sprint 4 class diagram for simviewer.jni

8.4.1.2. Class diagram for simviewer.data

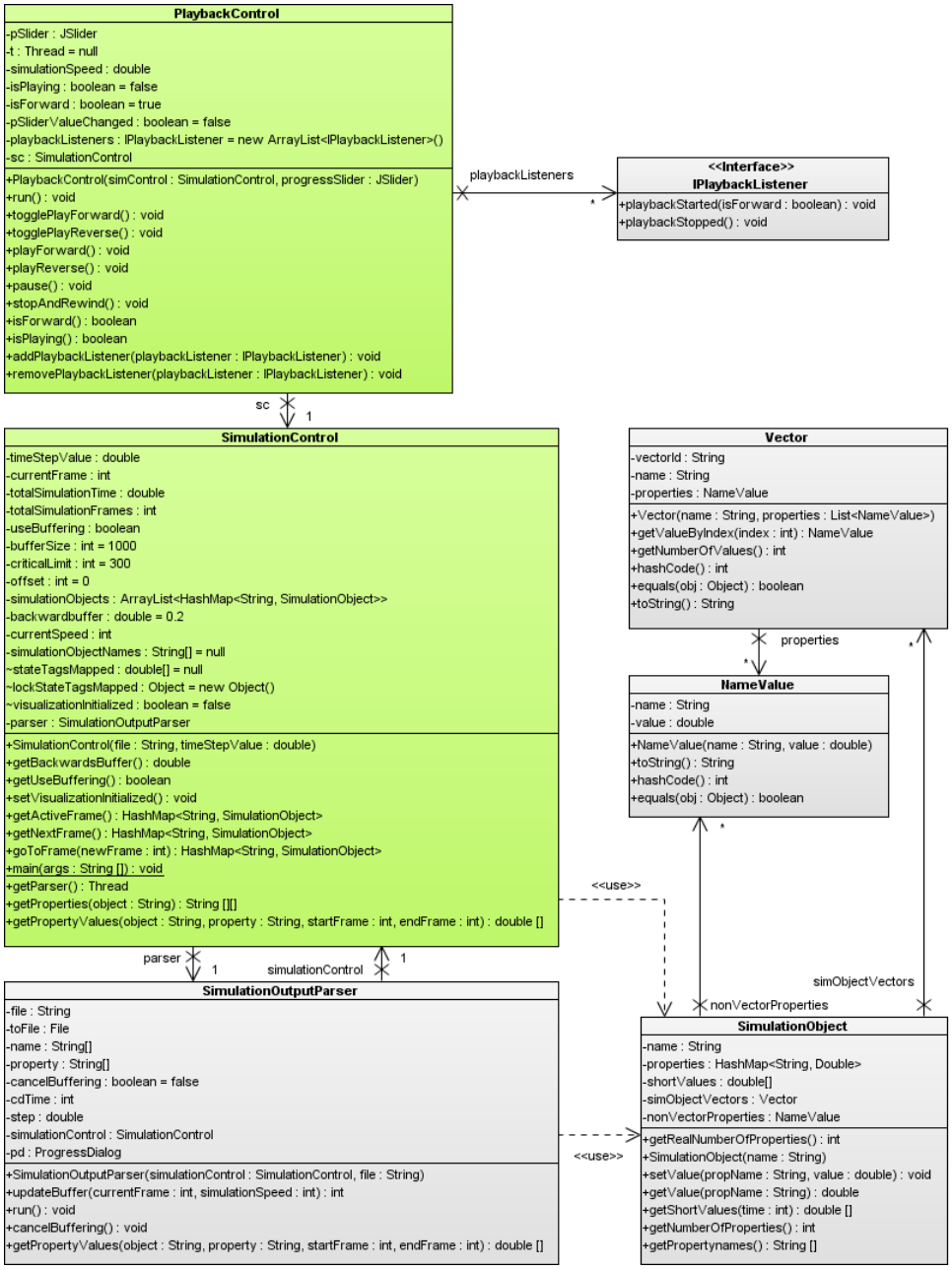


Figure 8.3: Sprint 4 class diagram for simviewer.data

8.4.1.3. Class diagram for simviewer.visualize

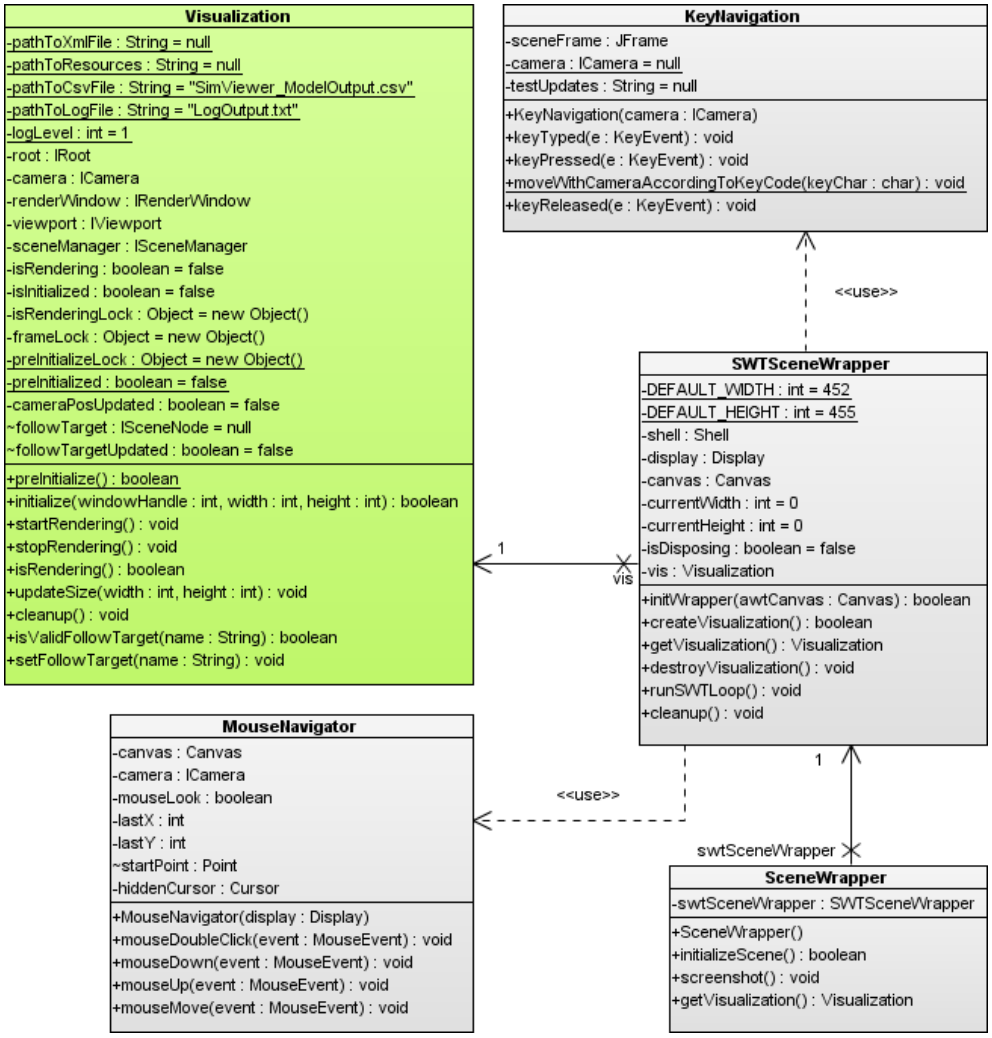


Figure 8.4: Sprint 4 class diagram for simviewer.visualize

8.4.1.4. Class diagram for simviewer.gui part 1

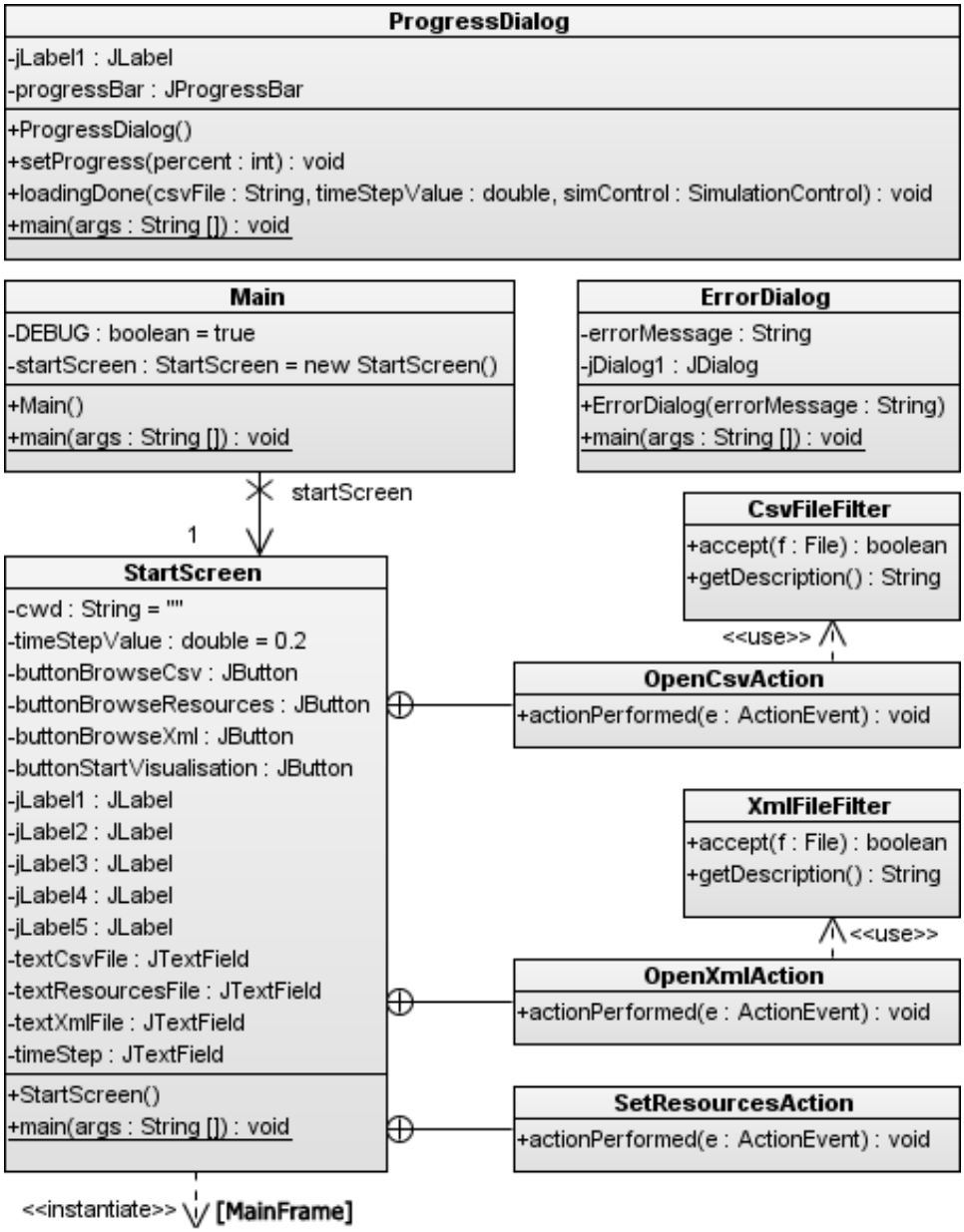


Figure 8.5: Sprint 4 class diagram for simviewer.gui part 1

8.4.1.5. Class diagram for simviewer.gui part 2

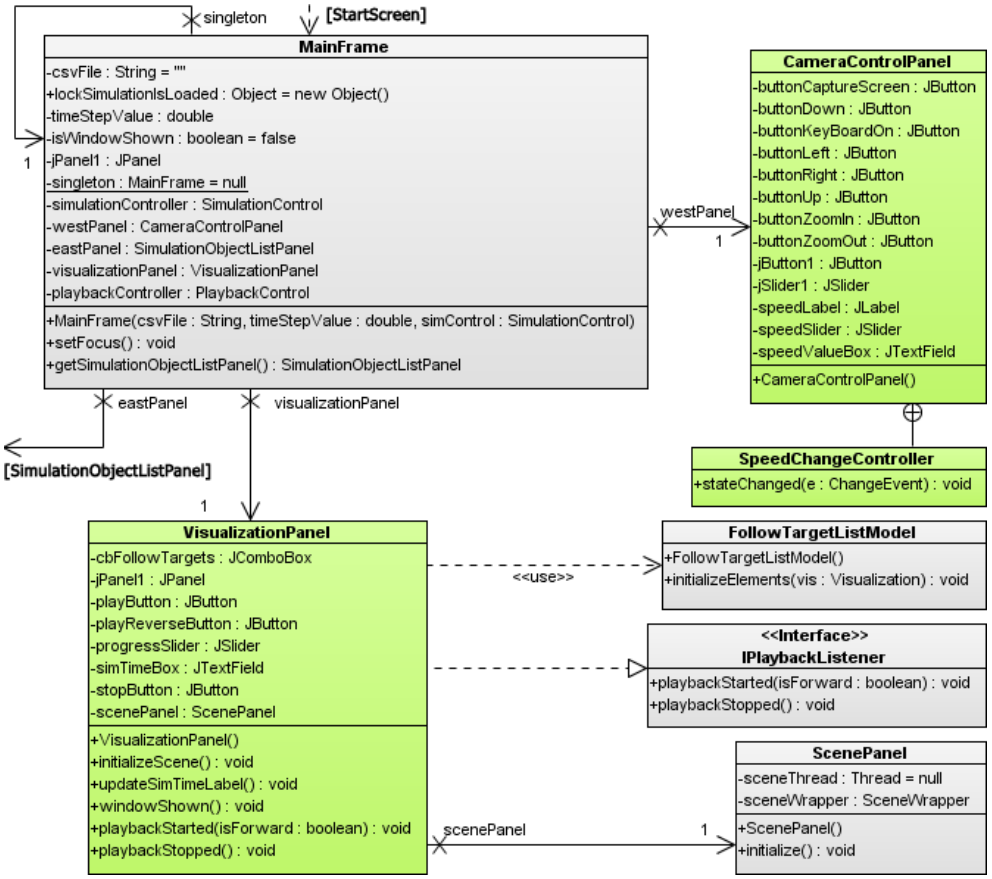


Figure 8.6: Sprint 4 class diagram for simviewer.gui part 2

8.4.1.6. Class diagram for simviewer.gui part 3

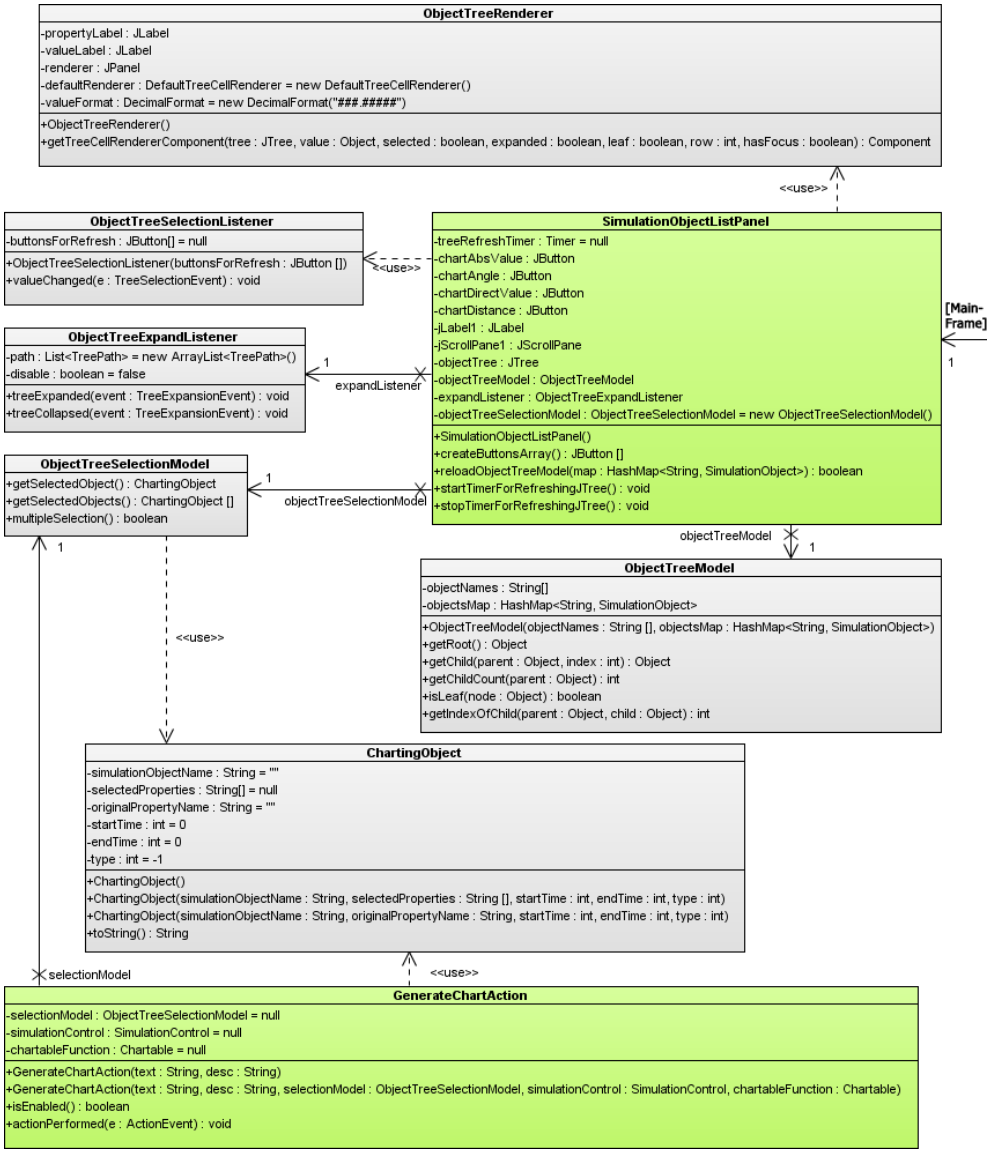

ObjectTreeRenderer
-propertyLabel : JLabel
-valueLabel : JLabel
-renderer : JPanel
-defaultRenderer : DefaultTreeCellRenderer = new DefaultTreeCellRenderer()
-valueFormat : DecimalFormat = new DecimalFormat("### .#####")
+ObjectTreeRenderer()
+getTreeCellRendererComponent(tree : JTree, value : Object, selected : boolean, expanded : boolean, leaf : boolean, row : int, hasFocus : boolean) : Component
<<use>>
ObjectTreeSelectionListener
-buttonsForRefresh : JButton[] = null
+ObjectTreeSelectionListener(buttonsForRefresh : JButton [])
+valueChanged(e : TreeSelectionEvent) : void
<<use>>
SimulationObjectListPanel
-treeRefreshTimer : Timer = null
-chartAbsValue : JButton
-chartAngle : JButton
-chartDirectValue : JButton
-chartDistance : JButton
-jLabel1 : JLabel
-jScrollPane1 : JScrollPane
-objectTree : JTree
-objectTreeModel : ObjectTreeModel
-expandListener : ObjectTreeExpandListener
-objectTreeSelectionModel : ObjectTreeSelectionModel = new ObjectTreeSelectionModel()
+SimulationObjectListPanel()
+createButtonsArray() : JButton []
+reloadObjectTreeModel(map : HashMap<String, SimulationObject>) : boolean
+startTimerForRefreshingJTree() : void
+stopTimerForRefreshingJTree() : void
[Main-Frame]
1
ObjectTreeExpandListener
-path : List<TreePath> = new ArrayList<TreePath>()
-disable : boolean = false
+treeExpanded(event : TreeExpansionEvent) : void
+treeCollapsed(event : TreeExpansionEvent) : void
1
expandListener
ObjectTreeSelectionModel
+getSelectedObject() : ChartingObject
+getSelectedObjects() : ChartingObject []
+multipleSelection() : boolean
1
objectTreeSelectionModel
objectTreeModel
1
ObjectTreeModel
-objectNames : String[]
-objectsMap : HashMap<String, SimulationObject>
+ObjectTreeModel(objectNames : String [], objectsMap : HashMap<String, SimulationObject>)
+getRoot() : Object
+getChild(parent : Object, index : int) : Object
+getChildCount(parent : Object) : int
+isLeaf(node : Object) : boolean
+getIndexOfChild(parent : Object, child : Object) : int
1
<<use>>
ChartingObject
-simulationObjectName : String = ""
-selectedProperties : String[] = null
-originalPropertyName : String = ""
-startTime : int = 0
-endTime : int = 0
-type : int = -1
+ChartingObject()
+ChartingObject(simulationObjectName : String, selectedProperties : String [], startTime : int, endTime : int, type : int)
+ChartingObject(simulationObjectName : String, originalPropertyName : String, startTime : int, endTime : int, type : int)
+toString() : String
selectionModel
<<use>>
GenerateChartAction
-selectionModel : ObjectTreeSelectionModel = null
-simulationControl : SimulationControl = null
-chartableFunction : Chartable = null
+GenerateChartAction(text : String, desc : String)
+GenerateChartAction(text : String, desc : String, selectionModel : ObjectTreeSelectionModel, simulationControl : SimulationControl, chartableFunction : Chartable)
+isEnabled() : boolean
+actionPerformed(e : ActionEvent) : void

Figure 8.7: Sprint 4 class diagram for simviewer.gui part 3

8.4.1.7. Class diagram for simviewer.chart.data

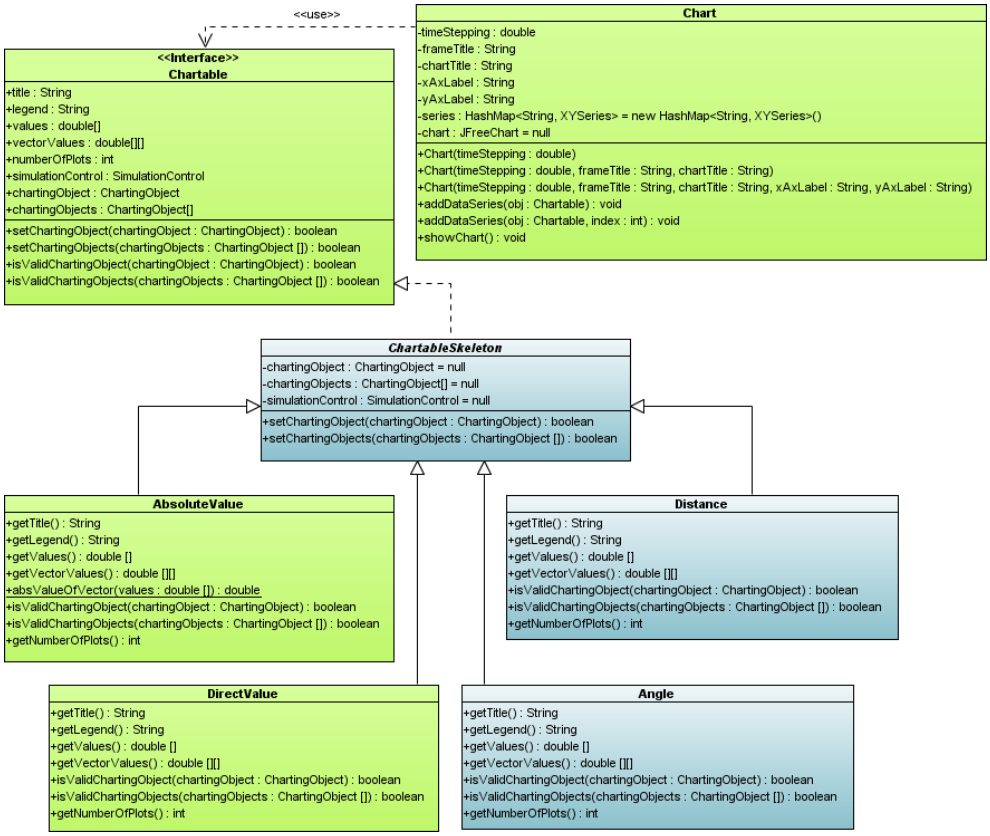


Figure 8.8: Sprint 4 class diagram for simviewer.chart.data

8.4.1.8. Class descriptions

8.4.1.8.a simviewer.chart.data

Class name	Description
ChartableSkeleton	An abstract class containing the common functionality of the charting functions.
Distance	Implementation of Chartable for calculating distance between two vectors.
Angle	Implementation of Chartable for calculating angle between two vectors, or three points.

Table 8.2: Sprint 4 class descriptions

8.5. Development

8.5.1. Changes in charting framework

This section describes all the changes in the charting framework that we did during sprint 4. These changes were necessary to fulfill the customer's needs.

8.5.1.1. Task definition

The task was split into several parts. First we needed to create new functionality in the charting framework which allows the user to create charts for multiple selected objects. Then we changed the behavior of the direct print function from the previous sprint. The direct print should print all the vector components in one chart. After that we added support for functions with more than one parameter.

The second part of the task was to implement two example functions with more than one parameter. The most important ones for the customer were:

- Angle between vectors – this function takes two or three vectors as a parameter and calculates the angle between them. In a situation with 3 vectors the middle one is taken as the base vector.
- Distance between vectors – calculate the distance between two vectors selected from the list.

8.5.1.2. Solution

The main problem of the solution from sprint 3 was that the chartable interface accepts only one ChartingObject. To plot multiple objects (multiple selections in the list) they had to be passed to the charting interface one at a time. This is a bottle neck of the previous solution and we decided to change it in this sprint.

The new approach to this problem was storing the reference to the array of selected objects in the chartable function and adding the function for validating the whole array of objects. This makes manipulation with multiple selections in the charting framework much easier. It also enables better support for functions that need more than one input. So new functions can use this array and get proper inputs for calculations from the array. They can expect proper values in the array because of the validating function which is called before the function could be used.

The behavior of the direct plot of a vector was changed very easily. We changed the interface by adding the method getVectorValues() .This method is used every time the charting object which is plotted in a chart is a vector. It allows the class which implements the Chartable interface to return a two dimensional array which is interpreted like multiple series in the chart.

When we finished all the necessary improvements for the application to be able to plot data correctly we figured out some other improvements. We realized that almost all the classes implementing the Chartable interface had some identical methods. This fact gave us the idea that we could improve the charting framework by adding the new abstract class ChartableSkeleton. This class implements all identical methods and provides a base which could be extended by other classes for charting. These classes previously implemented the Chartable interface.

8.5.1.2.a Implementation of new functions

During this process all the improvements described above were taken into account and we created new classes which are extending the ChartableSkeleton. First we created a class for calculating the Angle between vectors. The biggest problem was the implementation of the function which returns values for plotting data to the chart.

This function needed to make some calculations between vectors and their components. For solving this problem the SimViewer reads all the data from the input file into the temporary structure (temporary arrays)

and then makes the calculation between the values in each time step (for every column in the array) according to the formula for calculation of the angle between vectors. As the last step the application saves the result into another array which is returned as the result of this function.

The other parts of the interface were created very quickly according to the description of methods in the chartable interface. The distance function development has almost the same flow of actions. The only thing which was changed was the formula for calculating the distance, the other ideas are similar to the angle function.

8.5.1.3. Encountered problems

The most significant problem was changing the concept for multiple selections, because we didn't want to make many changes. We would like to keep the interface as simple as possible. Finally we found a way to manage multiple selections, but we changed the interface and we needed to add some functionality to the previous classes to be able to change the behavior of the direct vector plotting etc. On the other hand this version is prepared for almost every function which could be needed.

8.5.1.4. Conclusion

In this part of the development we changed the framework by editing the Chartable interface which caused some problems with adding some methods to the previously created classes (AbsoluteValue and Direct-Value), but in the end it will provide full support for multiple selections and for charting functions between vectors (generally between more values instead of functions with only one parameter). This makes the charting framework fully usable for the customer and it provides all necessary functions for fulfilling the requirement about charting.

We also improved the design a bit by adding an abstract class which simplifies the other classes.

8.5.2. *SimulationControl ↔ PlaybackControl*

In this sprint we fixed a rather big design issue from earlier sprints in both the simulation controller and the playback controller. We have so far had the speed control in the simulation controller, which is bad, because the speed affects how often we need to render, and as such the timing between frames. This timing is done in the playback controller, and should be there.

The speed control from earlier did not work in a proper way either, it jumped a number of frames according to speed instead of changing the time between each frame advance.

Moving the speed control led to quite major changes in the controllers due to the need of changing both the timing, what speed is chosen, whether to go forwards or backwards etc.

To implement this we made a speed variable in the playback controller. This combined with the time stepping and length of time used to render previous frame, is used to calculate timing between consecutive frames. The method performing the timing is generic, and performs as many fps as allowed by the system it runs on, according to the set parameters.

We also fixed a bug in which the application failed to compute the correct total simulation time. Before this was done by multiplying the total number of frames with the time step, but it should be total number of frames minus one multiplied with the time step. The reason for this is that the frame for the initialization is also in the total number of frames and should not be counted.

Additionally, a bug where the slider allowed going one frame too far was fixed in the Visualization pane. The application now stops when reaching the end of simulation, or reaching the start of simulation when going backwards, which is the desired behavior.

8.5.3. GUI changes

The speed controller was updated. It was changed to let the first ten values be 0.1, 0.2 up to 1.0, and then the next nine values are 2.0, 3.0 up to 10.0. Additionally we made a label that specifies to the user what speed he has chosen at all times. This label is updated while moving the speed slider.

Another function we added to the GUI was an overview of the simulation time. This gives the user information about where he is in the simulation time wise and how long the simulation is. The format used for these times are mm:ss.cc, where mm are minutes, ss are seconds and cc are partial seconds.

An update of this field is done upon each frame change, when sliding the simulation time slider and upon start of the program (in order to get the total simulation time set during initialization).

8.6. Testing

ID/Title	1.0 Testing of speed control
Description	In this test we want to test whether the functionality for adjusting speed during the simulation works correctly. We shall be able to adjust the speed of the playback both up and down.
Pre-condition	Initialized simulation, and in play mode.
Post-condition	Initialized simulation, in play mode, but with a different playback speed
Comments	
Pass	Ok

ID/Title	2.0 Testing of time information
Description	We want to test the information displaying at what time in the simulation file you currently are. To test this we will change the speed and jump in time several times to see if the information is still correct.
Pre-condition	Initialized simulation
Post-condition	Initialized simulation
Comments	
Pass	Ok

ID/Title	3.0 Charting of distance
Description	The user shall be able to select two vectors from the object list and chart the distance between these vectors during the simulation. The chart shall be displayed in a separate window.
Pre-condition	Initialized simulation.
Post-condition	Initialized simulation with new chart window.
Comments	
Pass	Ok

ID/Title	4.0 Charting of angle
Description	The user shall be able to select two or three vectors from the object list and chart the angle between these vectors during the simulation. The chart shall be displayed in a separate window.
Pre-condition	Initialized simulation.
Post-condition	Initialized simulation with new chart window.
Comments	
Pass	Ok

8.7. Conclusion

In the last sprint there weren't much non-optional functionality left to implement. The main task was to add more functions for charting, like distance between objects and angle between vectors.

We also had to solve the problems with time stepping from last sprint and do a lot of testing and bug fixing.

We also improved the GUI by creating icons instead of buttons so the scene can be shown in a bigger frame.

The rest of the time we used to prepare our final presentation and finish the report.

Chapter 9
After Work

9.1. Chapter introduction

This chapter describes the process during the final days before report delivery and presentation.

This phase took place between November 14th and November 20th, which includes the last weekend prior to delivery.

9.2. Project report

During the after work, the project report had to be finalized.

This included adding and completing the documentation for the last two sprints and the after work, evaluating the project, writing a user guide, proof reading and ensuring that the project report was consistent.

Overview of time spent in the whole project was added in Appendix B.

9.3. Presentation

During the after work, the final presentation was prepared.

Presentation slides with diagrams and application demo was created. The presentation was rehearsed several times, both internally in the group, as well as for the supervisors.

Chapter 10

User Guide

10.1. Chapter introduction

This chapter describes how to use the SimViewer application.

Since the application is only a prototype and therefore wasn't required to have an installation program, there is no documentation about this. It will be left to the customer to provide the application to their clients in a sensible way with instructions on how to get it running.

10.2. System requirements

10.2.1. Operating System

SimViewer has been tested and is known to work on Microsoft Windows XP and Microsoft Windows Vista.

10.2.2. Software prerequisites

SimViewer uses libraries that require the Microsoft Visual C++ 2005 SP1 redistributable package. This must be installed on the system prior to running the application. This package can be found on Microsoft's web pages[9].

The Java SE Runtime Environment must also be installed. SimViewer require version 6 or later, which must be installed on the system prior to running the application.

[9] http://www.microsoft.com/downloads/details.aspx?familyid=200B2FD9-AE1A-4A14-984D-389C36F85647&displaylang=en

The required JRE package can be downloaded from Sun Microsystems' web pages[10].

10.2.3. Hardware requirements

SimViewer's hardware requirements are dependent on the simulation configuration loaded, and can thus not be specified in general. The following requirements must however be fulfilled independent of the simulation configuration:

- OpenGL compatible graphics card
- Approximately 15 MB of hard drive space, excluding configuration specific files
- Approximately 128 MB of available RAM

[10] http://java.sun.com/javase/downloads/?intcmp=1281

10.3. Startup dialog

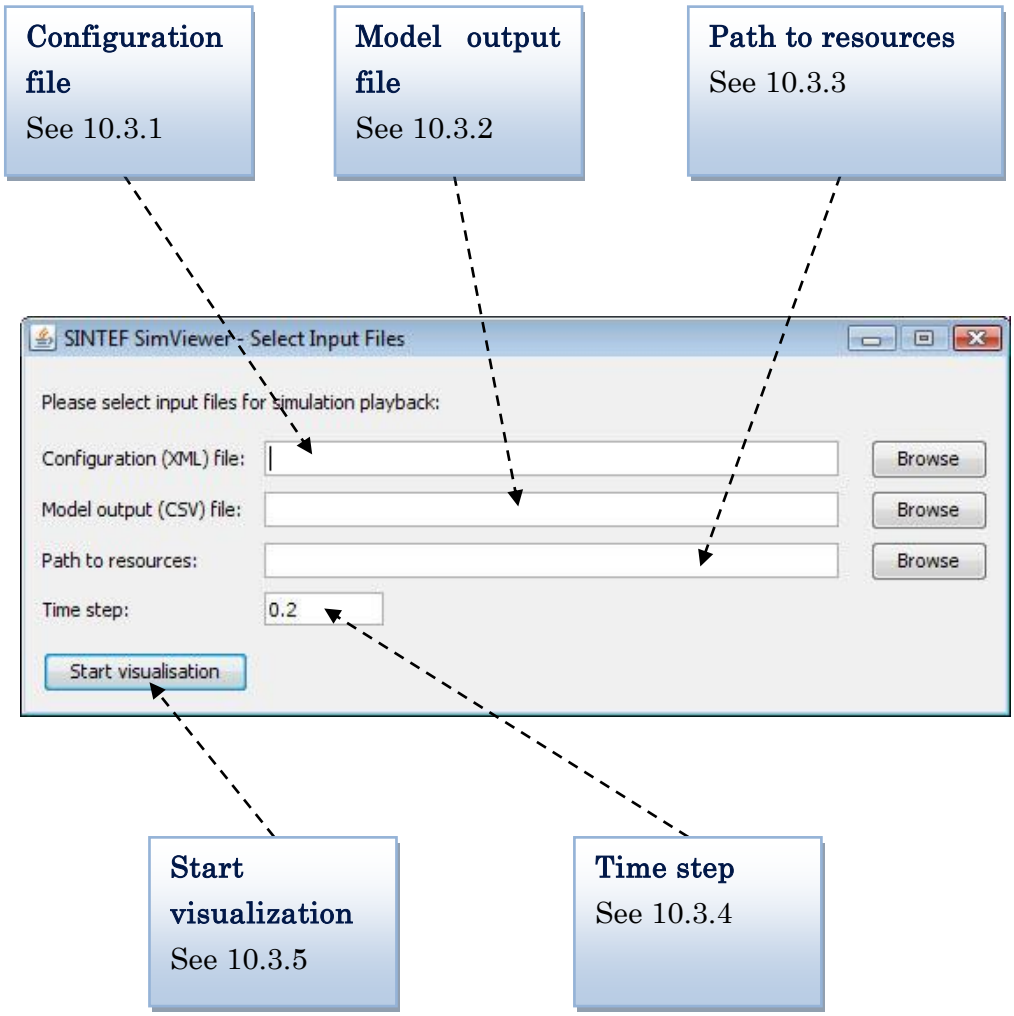


Figure 10.1: Startup dialog guide

10.3.1. Configuration file

Specify the path of the configuration XML file. Every simulation has one and it is related to the CSV file.

10.3.2. Model output file

Specify the path of the CSV file. This file is related to the configuration file (10.3.1).

10.3.3. Path to resources

Specify the path of the resources folder. This folder should be included with every simulation.

10.3.4. Time step

Specify the desired time step value. Refer to the CSV file for the time resolution. If you specify a time step value that does not match with the CSV file, interpolation will occur for time entries missing in the CSV file.

10.3.5. Start visualization

Press the button when all the fields have been correctly set. If some values are missing while button is pressed an error message will be displayed.

10.4. The simulation main window

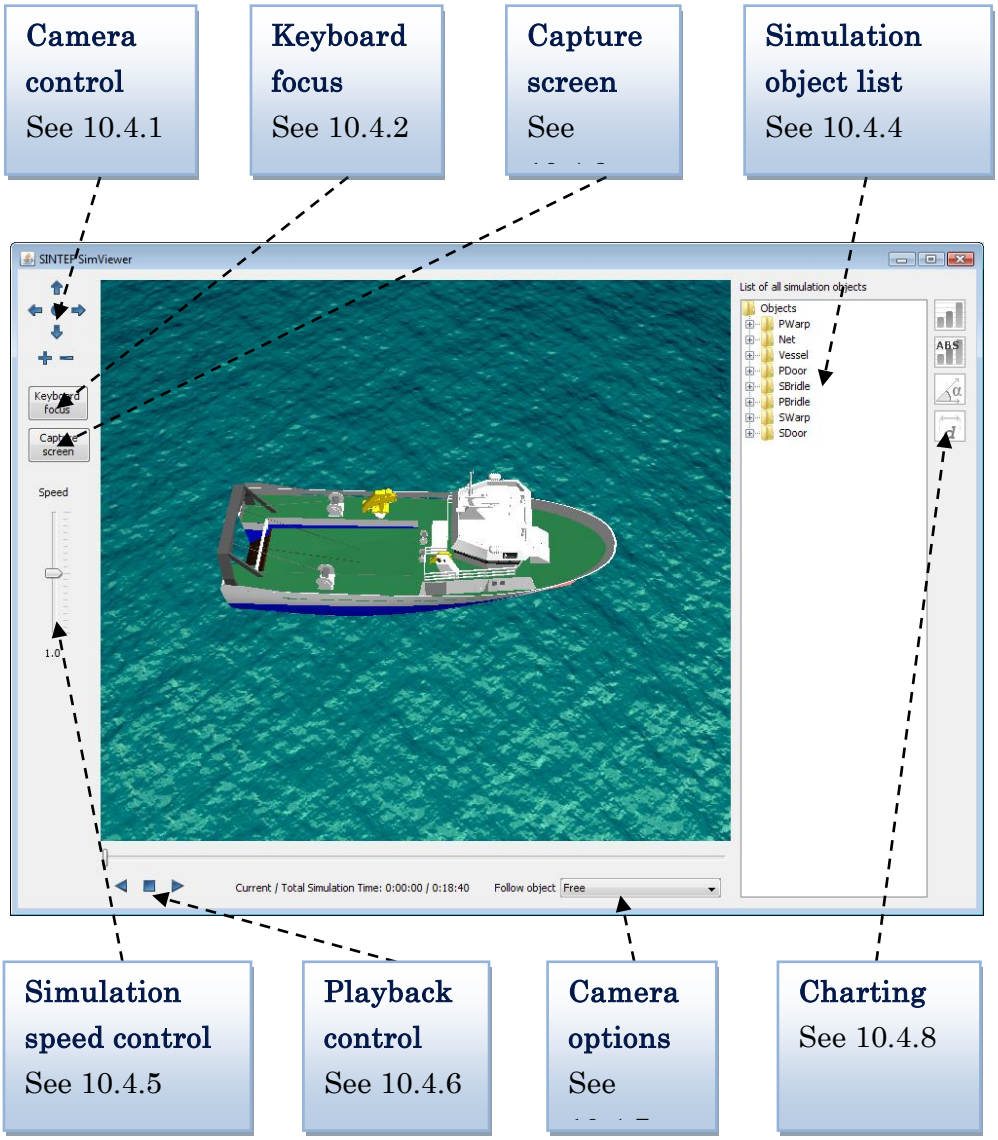


Figure 10.2: Main window guide

10.4.1. Camera control

Use the arrows to move the camera to your wanted position. The + and – buttons can be used to change the zoom of the camera.

10.4.2. Keyboard focus

When this button is pressed, keyboard focus is gained (if keyboard focus was not already active). It enables the shortcut keys to control the camera. One can then use the keyboard shortcuts instead of the camera control buttons.

Key	Action
Q	Zoom out
A	Zoom in
E	Move camera to the left
R	Move camera to the right
W	Move camera down
S	Move camera up

10.4.3. Capture screen

Use this button to capture the current simulation frame. When clicked, a dialog where one can specify directory to save screenshot will pop up.

10.4.4. Simulation object list

This list contains all the simulation objects and its properties and values. The list is organized as a hierarchy where the simulation object is the top node which can be expanded to show the objects properties. All vectors can be collapsed, and the property values are updated during playback. One can use this list to observe values change during the simulation.

10.4.5. Simulation speed control

Use this slider to control the simulation speed. One can increase the simulation speed by lifting the slider and decrease the speed by lowering the slider.

10.4.6. Playback control

The playback slider

The slider shows where in the simulation we are. Use the slider to skip frames or jump to specific places in the simulation. Note that when the slider is moved re-buffering may occur.

Play

The simulation starts playing from the current frame.

Backwards play

The simulation starts playing backwards from the current frame.

Pause

The simulation pauses at the current frame.

Stop

The simulation stops and current frame is set to the start of the simulation

10.4.7. Camera options

The camera is by default set not to follow any object (“Free”). This means the camera will not follow any objects and the user may freely control the camera. Sometimes it is handy to follow some specific object.

All the simulation objects in the current simulation also occur in the list where the user can select which object the camera should follow. By changing this to some other object than "Free", the camera will automatically be attached to this object and follow it until a new object is chosen.

10.4.8. Charting

There is possible to create four different charts. The user has to select the properties from the object list in order to use the different charting functions.

When a charting function is disabled, it is grey, and when it is enabled, the color is turned on. Functions are automatically enabled based on the selection in the simulation object tree.

Please see the table below for description of the different charting functions.

Disabled	Enabled	Description
		Chart the value of the selected vector(s) and/or vector component(s).
		Chart the absolute value of the selected vector(s) and/or vector component(s).
		Chart angle between two (treated as directions) or three (treated as positions) three dimensional vectors. Note that exactly two or three vectors have to be selected for this to work.
		Chart distance between two three dimensional vectors. Note that exactly two vectors have to be chosen for this to work.

Chapter 11
Evaluation

11.1. Chapter introduction

This chapter documents our evaluation of the project.

11.2. Internal process

During the project we have not had any clear leader. There have been many suitable leader candidates in the group, and we have all taken responsibilities when we have notices that something could be improved. In the start of the project we did not know each other very well. It therefore took us some time to get things going. The course in group dynamics was of much help in making us cooperating more efficient as a team. We learned to know each other and found our roles in the team.

We reached the project goal. This has been a test project (proof of concept), and the main goal for the customer was to see if this could work. Customer was pleased with the result after sprint 3. In the sprint review meeting they were very satisfied.

11.2.1. What we did well

During the project there have been *no personal conflicts*. All group members have felt like they could raise their voice and *speak their minds out loud*. It has been quite an informal and *relaxed environment* where *no clear leader has been dominating*. The structure of the team has been quite flat.

We have had *few disagreements* because we have been able to *communicate quite efficiently*. All problems have been solved because we have felt like we were not afraid to speak our minds, and all team members have been happy to contribute with their special competence and learn from the others.

The assignment *of work has been sensible and balanced* and we have not had any team members that contributed less than others. Members have shown interest in catching tasks they thought they could solve efficiently with their competence and experience. So everyone has been working with something they thought of as meaningful for themselves.

11.2.2. What we did not do well

The meetings were sometimes quite unstructured which resulted in *ineffective meetings*. Internal discussions within some small part of the group could arise, which lead to people not being part of this discussions getting bored. This was due to the fact that our team *did not have any clear leader*.

There was also a problem that *we did not prepare meeting agendas for group meetings* which resulted in the group members not being as prepared for the meetings as they could have been. The topics of the meetings were therefore sometimes chosen arbitrary.

We were *too quick with the start of the development phase*. We should have used more time resources to choose a design before we started to develop. This resulted in some waste of time because of some simplifications we made. This time could have been spent more judiciously. The reason for this, however, is the nature of the preliminary studies, which included coding of tests which were then used directly in the development phase.

During the project we have had *too little focus on documentation*. This resulted in a lot of after work. We should have used more time to write the documentation while we were developing. This would have saved a

lot of time resources because we would have everything in our minds while writing the documentation.

11.2.3. What we learned

During the course we've had a few more or less valuable lectures about different topics. We can point out a few of them as being especially valuable to us and the course:

- Group dynamics course: This course helped us a lot to learn how to work together as a team and to learn to know each other. We learned a lot about ourselves and our team members and it showed to be quite valuable during all phases of the project.
- Technical writing seminar: This seminar was quite valuable in the sense of how to structure formal reports, considering that few of us have had any prior experience with this.
- Presentation technique lecture: This lecture was very interesting. We learned how to proceed to our stress and nerves that occur for most of us in presentations. How we could structure presentations well, how to make presentations interesting, what kind of words and language to use (keep in mind who you are making the presentation for), and be very aware of how you use your time (you want to use it appropriate).

We have also learned how a project evolves from start to end. We have been through each phase of a real project from requirement specification, design, implementation to testing.

We have also seen that there often is conflicting views between how customer and the project group understands the requirements. In this context we have learned the one of the most valuable things about agile development methodologies. The fast feedback you get is really valuable and can help to recover a project from a faulty track.

We used Scrum as our agile approach, and we have gathered valuable experience with this approach that will be valuable in later software development projects.

We have also learned about project management tools. This includes tools for registration of working hours, tasks that need to be done and tools for monitoring overall project progress. Estimation of tasks and how valuable it is if one can be able to almost accurately estimate hours of each task has been one of our experiences.

Team cooperation skills in quite big team of 6 persons helped us learn how to communicate efficiently and how important it is to define clear responsibility roles and divide the work sensible. Our communication and writings skills in English have also improved significantly. Especially because two of our team members are from Czech Republic and we needed to communicate in English in order to understand each other.

We have also learned about how to relate to a real customer and problems that may occur in such a relationship (misunderstood requirements etc.). We have also learned to be more responsible in a team where every team member is dependent of your work and you yourself are not the only one who will have to pay if you don't do what you are intended to do. This has the same validity about the customer.

11.3. Customer

During the project, one of the members of the team was responsible for the contact with the customer. Our customer contact was Vegar Johansen throughout the project. All contact with SINTEF was in written English and it was made available for all team members with help from the team mailings list.

Halfway in the project two more persons from SINTEF were involved. Vegar was gaining more responsibility at SINTEF so he was not able to spend as much time on our project as he first intended. Karl Johan Reite and Martin Føre were therefore involved to be able to support

Vegar. We did, however, never notice any problems or resource problems at SINTEF after this change.

During the project, the communication with Vegar has been quick and effective. There have been no conflicts and all the issues regarding requirements specification have easily been solved by written or oral communication.

11.4. Supervisors

At the beginning of this course we think that we got too little feedback from the supervisors. They guided us a bit on the way, saying what we should do, but we didn't get too many comments on the work we did.

After the meeting with all the supervisors in the beginning of October the feedback was much better. We started to get feedback on things we did in the beginning of September, and we feel that it would have been much better if we could have gotten this feedback already in September. It was hard to start on something we thought we had finished.

However, the feedback we have gotten after October has been good. We have fixed a lot of things and we have therefore made a much better report.

We also feel that we need to comment one of our advisory meetings. Our main supervisor had told us that he was not able to show up, but we would be notified if the meeting had to be cancelled. All group members showed up for the weekly meeting as always, but not our assistant supervisor. After we had waited 10 minutes a person came and told us that he would not show up. We think this was a bit unprofessional and wasted a lot of time for our group.

There has also been a bit of changing information during this course. The supervisors didn't seem to be one hundred percent informed about

everything in the course. There have been several things we have asked them that they either didn't know or said things that were not correct or only partly correct. Other times they have disagreed with the information booklet and we felt that it was hard to relate to this conflicting information.

As a last point about the supervisors we would like to discuss the fact that our assistant supervisor went on a trip abroad three days before final report delivery and presentation. We were informed about this only a week before he was leaving.

This caused some problems for us, as we suddenly needed to finish the report earlier than planned so that the supervisors could read through it before they left. Also this meant we would not be able to get much feedback if we had any questions near the end.

We also needed to reschedule our final supervisor meeting. Considering our conflicting schedules, with a lot of other subjects having final deliveries and other important lectures, we used too much time on finding a suitable time. This was time we should have spent on the report instead.

11.5. Further work

We implemented all the non-optional requirements for this project. None of the optional requirements were completed due to time limitations.

However, the point of this project was to provide a program that had basic functionality to prove that the concept of the application was possible and useful. Therefore the functionality of this application can be extended by the customer at a later stage.

We have tried to make the code for the different parts extendible. For example the charting framework can easily be extended with additional charting functions.

There could be made a packaging of our application when distributing it to other clients. This could be either by packing it into a zip file completely set up for a specific configuration, or by making a custom installation program which installs it on a client computer.

11.5.1.1. Known issues

We have a known bug in the data part, where the parser stops responding when a large and/or sudden movement in time is performed. This only occurs when the loaded file must be buffered.

The cause of this is not known entirely, but we suspect that it is either because the thread itself has crashed, or because of a deadlock in interacting with the simulation controller. There are several parts of the code between these two classes that require synchronization, which make deadlocks possible (each class/thread is waiting for the other to return a result, and as such never continues).

For now, we have a default, but rather large, buffer size, and it does not currently need buffering for the example files provided. With this configuration it works perfectly. It will, though, enable buffering if the simulation file exceeds 1 000 lines of data.

11.6. Suggestions for improvements

This section describes our main ideas on how to improve the course.

11.6.1. Lecture schedule

We strongly suggest that there should be a better time schedule for the lectures. Most of the lectures were given too late to be useful. An example is the use case estimation lecture on 30th September. We had to be done with pre-delivery before this lecture, which needed to contain an initial requirement specification.

Another problem was that the lectures didn't respect the other courses. There should be more hours reserved for this course in the time schedule of NTNU. Customer driven project has only reserved two hours per week for this semester, but in reality it required much more. This was especially the case in the beginning of the semester, where there were collisions with other subjects.

There were also problems with sending information about cancelled lectures. Especially the Scrum lecture was cancelled without prior notification by e-mail. No one came to the lecture and we received the message about the cancelling of this lecture the next day. This was an unprofessional approach.

On the other hand, we can say that there were some really useful lectures. We all liked the group dynamics seminars, although they were a bit long, but in our opinion they could be also given earlier. The presentation technique lecture gave us some really good ideas of how to prepare the presentation, and this lecture could also be given earlier.

Additionally, it would be better to start from the very beginning of the semester to get one more week for the project.

11.6.2. Information flow

We feel that there were a lot of misunderstandings during the course. The students should have gotten better information about available resources (e.g. offices in P-15) as soon as possible – probably during the first lecture or in the info booklet.

There should also have been more suggestions and guidelines to help make the critical startup of the project more efficient.

This issue is also connected with the school controlling customer materials and ideas. We suggest that the supervisors should meet the customer at least a week before the first meeting with the students. This meeting (customer and supervisor) would provide a good chance to check if the customer has a clear idea of what he wants and if he has prepared all the material the students need to get the project started.

The information about the grading was also really confusing. It was mentioned in the course materials but when we asked during the first supervisor meetings, we got totally conflicting information. The students should have a clear definition of grading when the course starts and all other participants (supervisors, course leaders) should have and convey the same information.

11.6.3. Supervisors

We think that the supervisors should have more meetings with the course leaders for them to be able to provide better and consistent information to the students. Basically we can say that they should be better informed.

11.6.4. Scrum vs. waterfall

The course allowed the students to use the Scrum method of development, but the course and its structure didn't support it. We followed Scrum where the requirements can be changed during development, but we needed to deliver final and fixed requirement specification in pre-delivery (according to supervisors).

We also needed to document everything in a traditional way, because according to the course material we needed to have a 200 pages long report. The information about how many pages we needed was also conflicting.

These limitations make Scrum more difficult and less efficient. This course should definitely have different requirements on the report structure for waterfall and Scrum models.

11.6.5. Assigning students

The last improvement could be some kind of pre-registering. It would be nice to have the opportunity to sign up for specific projects online. It would save a lot of time in the first lecture and it also gives the students possibility to choose a project which is suitable for them.

In the current solution someone could get a project oriented on a topic completely different from what one has studied before (different programming language, different specialization). This subject should simulate a real world situation and in the real world you would get a task where at least some of your skills are relevant.

We do, however, realize that the randomness ensures an even distribution of the students, and we liked the fact that we had a group where no one knew each other from before.

Glossary

3D: Three dimensional

API: Application Programming Interface.

AVI: Audio Video Interleave. A video file format wrapper that allows for both video and audio streams interleaved.

BSD License: Berkeley Software Distribution License. A free license of permissive nature.

C: A procedural programming language.

C++: A C-style programming language with object-orientation.

CPU: Computer Processing Unit.

CSV: Comma Separated Values

CVS: Concurrent Versions System. A version control system developed by Dick Grune.

DLL: Dynamic-link Library. Microsoft's implementation of the shared library concept.

EPS: Encapsulated PostScript. File format used to describe images and drawings.

eRoom: A SINTEF provided project management tool for handling documents.

fhSim: A SINTEF created library for calculating simulations

fhVis: A SINTEF created library that extends fhSim with visualization capabilities.

GNU: GNU's Not Unix. A free software, mass collaboration project.

GPL: General Public License. A license which requires all derivative work to be published under the same license.

GUI: Graphical User Interface.

HTTP: Hypertext Transfer Protocol.

IDI: Department of Computer and Information Science

Java: Object-oriented programming language developed by Sun Microsystems

JavaBean: A reusable software component for Java.

JNI: Java Native Interface. Enables Java to communicate with native code.

JPEG: Joint Photographic Experts Group. An image file format with lossable compression.

JVM: Java Virtual Machine. An environment that interprets and executes Java programs.

LGPL: Lesser General Public License. A license allowing for commercial use.

Library: In software development, refers to a collection of functionality that can be used modularly.

MPEG: Moving Picture Experts Group. A video format for compressed video and audio.

Ogre4j: An Java/JNI port of OGRE

PDF: Portable Document Format. File format created by Adobe Systems for document exchange.

PNG: Portable Network Grapic. An image format with lossless compression.

RAM: Random Access Memory. Often the type of working memory of a computer.

Singleton: A paradigm where a class can only be instantiated once.

SINTEF: The Foundation for Scientific and Industrial Research

SSH: Secure Shell

SVG: Scalable Vector Graphics. File format for describing images and drawings that can be resized without information losses.

SVN: Subversion. A version control system initiated by CollabNet.

UML: Unified Modeling Language. A general-purpose modeling language in the field of software engineering.

UTF: Unicode Transformation Format. A set of formats for binary text representation.

Vista: Microsoft Windows Vista. An operating system in the Windows series

XML: eXtended Markup Language

XP: Microsoft Windows XP. An operating system in the Windows series

References

1. Ogre4j. [Online] [Cited: September 18, 2008.] http://www.ogre4j.org.

2. **Collaboration.** Standard Widget Toolkit. *Wikipedia.* [Online] 09 13, 2008. [Cited: 09 14, 2008.] http://en.wikipedia.org/wiki/Standard_Widget_Toolkit.

3. —. Swing (Java). *Wikipedia.* [Online] September 1, 2008. [Cited: September 14, 2008.] http://en.wikipedia.org/wiki/Swing_(Java).

4. **Strenn, Stephen.** Swing vs. SWT Performance: Have a Look at the Call Stacks. *Java Lobby.* [Online] March 3, 2006. [Cited: September 14, 2008.] http://www.javalobby.org/java/forums/t65168.html.

5. **Marinilli, Mauro.** Swing and SWT: A Tale of Two Java GUI TA \s "GUI" Libraries. *developer.com.* [Online] April 9, 2003. [Cited: September 14, 2008.] http://www.developer.com/java/other/article.php/10936_2179061_1.

6. **Hirsch, Gordon.** Swing/SWT. *Eclipse.org.* [Online] June 20, 2007. [Cited: September 15, 2008.] http://www.eclipse.org/articles/article.php?file=Article-Swing-SWT-Integration/index.html.

7. JFreeChart. [Online] [Cited: September 17, 2008.] http://www.jfree.org/jfreechart/index.html.

8. JCCKit. [Online] [Cited: September 17, 2008.] http://jcckit.sourceforge.net.

9. QN Plot. [Online] [Cited: September 17, 2008.] http://quies.net/java/math/plot/.

10. OpenChart2. [Online] [Cited: September 17, 2008.] http://approximatrix.com/products/openchart2/.

11. PT Plot. [Online] [Cited: September 17, 2008.] http://ptolemy.eecs.berkeley.edu/java/ptplot5.1p1/ptolemy/plot/doc/index.htm.

12. JRobin. [Online] [Cited: September 17, 2008.] http://www.jrobin.org/.

13. JOpenChart. [Online] [Cited: September 17, 2008.] http://jopenchart.sourceforge.net/.

14. jCharts. [Online] [Cited: September 17, 2008.] http://jcharts.sourceforge.net/.

15. JChart2D. [Online] [Cited: September 17, 2008.] http://jchart2d.sourceforge.net/index.shtml.

16. Chart2D. [Online] [Cited: September 17, 2008.] http://sourceforge.net/projects/chart2d.

17. ThunderGraph. [Online] [Cited: September 17, 2008.] http://sourceforge.net/projects/thundergraph.

18. E-Gantt. [Online] [Cited: September 17, 2008.] http://sourceforge.net/projects/jgantt.

19. MagPlot. [Online] [Cited: September 17, 2008.] https://magplot.dev.java.net.

Appendix A

Involved parties

A.1. Project team

Erling Børresen	erlingbo@stud.ntnu.no
Pavel Fatrdla	pfatrdla@seznam.cz
Erik Hidle	erikhi@stud.ntnu.no
Jiří Kadlec	jir.kadlec@post.cz
Rolf Klungsøyr	klungsoy@stud.ntnu.no
Jørgen Nystad	jorgenys@stud.ntnu.no

A.2. Customer

SINTEF Fisheries and Aquaculture, represented by:

Vegar Johansen	vegar.johansen@sintef.no
Karl Johan Reite	karl.j.reite@sintef.no
Martin Føre	martin.fore@sintef.no

A.3. Supervisors

Sundar Gopalakrishnan	sundar@idi.ntnu.no
Basit A. Khan	basit@idi.ntnu.no

Appendix B

Project Plan Resources

B.1. Time statistics

B.1.1. Preliminary study

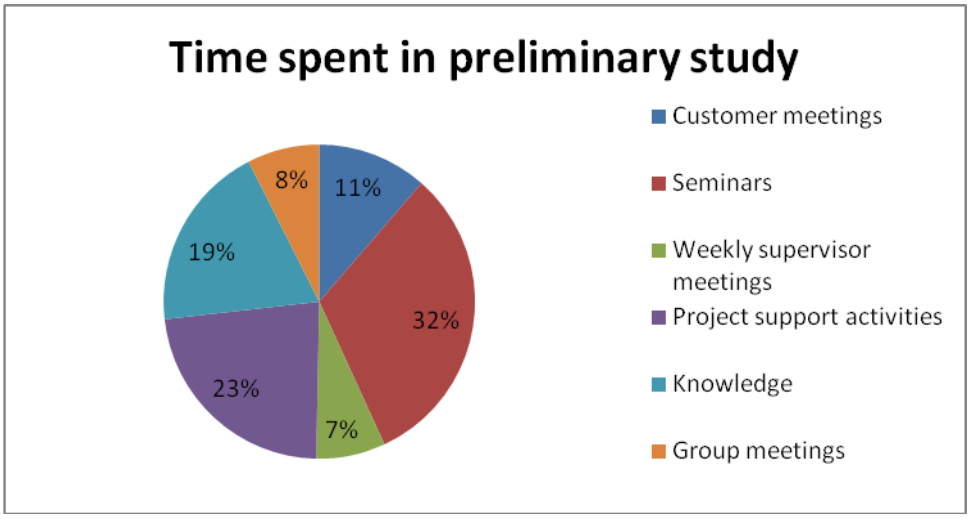


Figure B.1: Preliminary study time distribution

Stories ⊕ Tasks	Estimated hours	Actual hours
Group meetings	25	24
Weekly supervisor meetings	25	23
Customer meetings	40	36.5
Seminars	110	102
Project support activities	85	73.5
⊕ Project report writing		
⊕ Requirement specification		
⊕ Risk management		
⊕ Make use cases		
⊕ Project plan		
⊕ Write meeting minutes		

Knowledge	75	62
⊕ Java charting framework		
⊕ Ogre4j		
⊕ fhVis		
Summary	**360**	**321**

B.1.2. Sprint 1

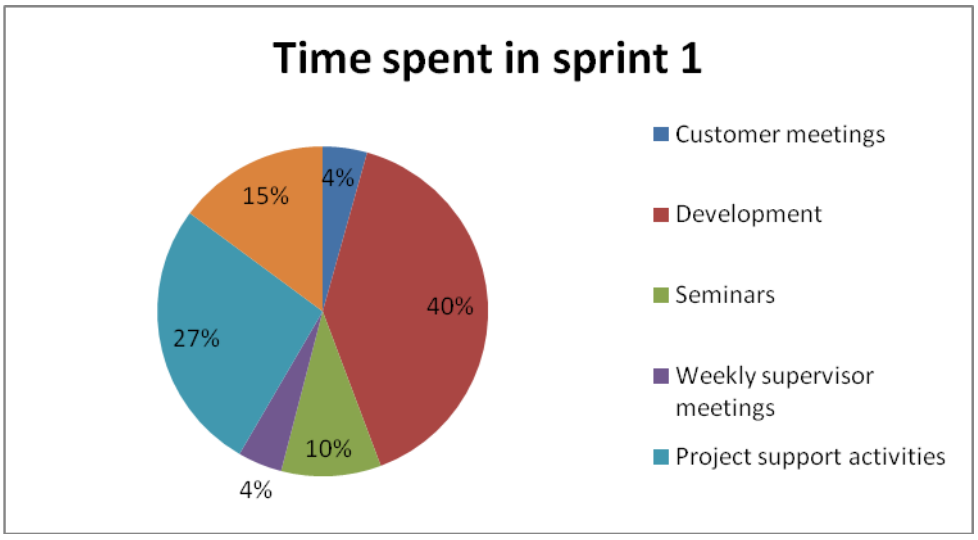


Figure B.2: Sprint 1 time distribution

Stories ⊕ **Tasks**	**Estimated hours**	**Actual hours**
Group meetings	43	41
Weekly supervisor meetings	12	12
Customer meetings	15	12
⊕ Demonstration/Acceptance		
Development	120	110
⊕ Design		
⊕ GUI		
⊕ Showing 3D scene in Swing		
⊕ Reading configuration file		
⊕ Parsing simulation output file		

Stories ⊕ Tasks	Estimated hours	Actual hours
Seminars	30	27
Project support activities	80	73.6
⊕ Project report writing		
⊕ Sprint backlog		
⊕ Gant diagram		
⊕ Weekly report		
⊕ Write meeting minutes		
Summary	**300**	**275.6**

B.1.3. Sprint 2

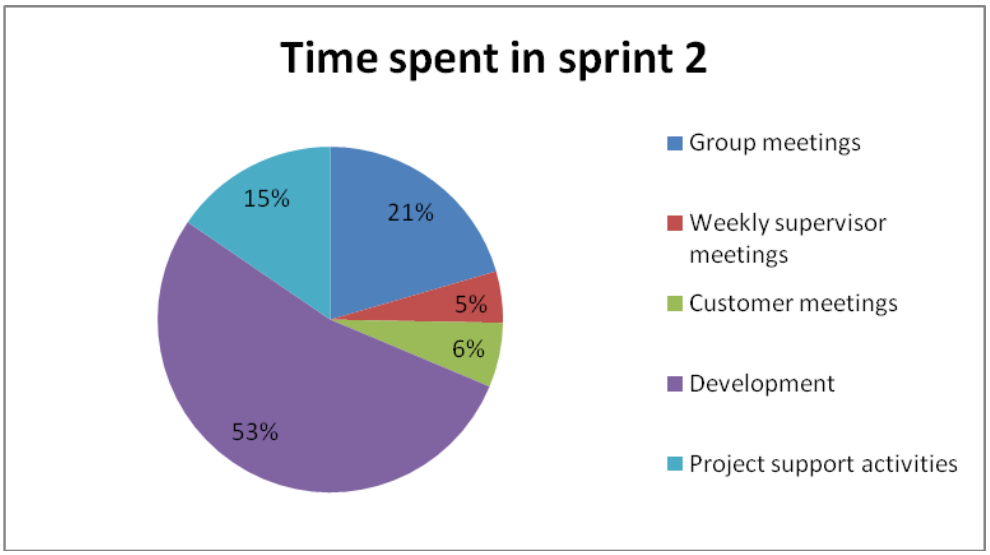


Figure B.3: Sprint 2 time distribution

Stories ⊕ Tasks	Estimated hours	Actual hours
Group meetings	50	51
Weekly supervisor meetings	12	12
Customer meetings	15	15
⊕ Demonstration/Acceptance		

Stories ⊕ Tasks	Estimated hours	Actual hours
Development	120	132.5
⊕ List of objects in JTree + updating		
⊕ Parsing of output file with interpolation		
⊕ Initial playback controller		
⊕ Fixing Vista issues with fhVis		
Project support activities	43	38.5
⊕ Project report writing		
⊕ Sprint backlog		
⊕ Class diagrams		
⊕ Weekly report		
⊕ Write meeting minutes		
Summary	**240**	**249**

B.1.4. Sprint 3

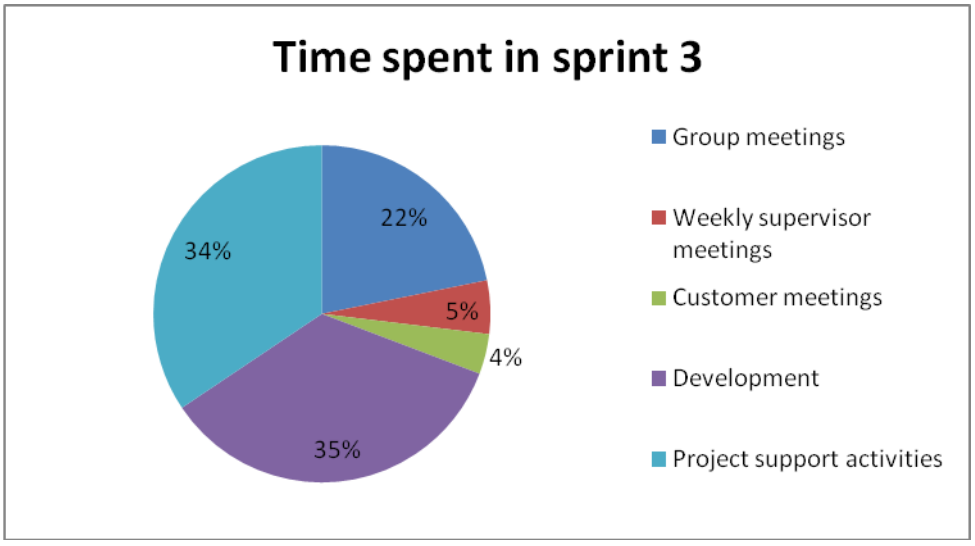


Figure B.4: Sprint 3 time distribution

Stories ⊕ Tasks	Estimated hours	Actual hours
Group meetings	50	51
Weekly supervisor meetings	12	12

Stories ⊕ Tasks	Estimated hours	Actual hours
Customer meetings	15	9
⊕ Demonstration/Acceptance		
Development	120	81.5
⊕ Object tree selection values and passing to charting framework		
⊕ Work on the data package		
⊕ Screenshot		
⊕ Charting		
⊕ Improving GUI		
⊕ Following camera		
Project support activities	43	80.5
⊕ Project report writing		
⊕ Sprint backlog		
⊕ Class diagrams		
⊕ Weekly report		
⊕ Write meeting minutes		
Summary	**240**	**249**

B.1.5. Sprint 4

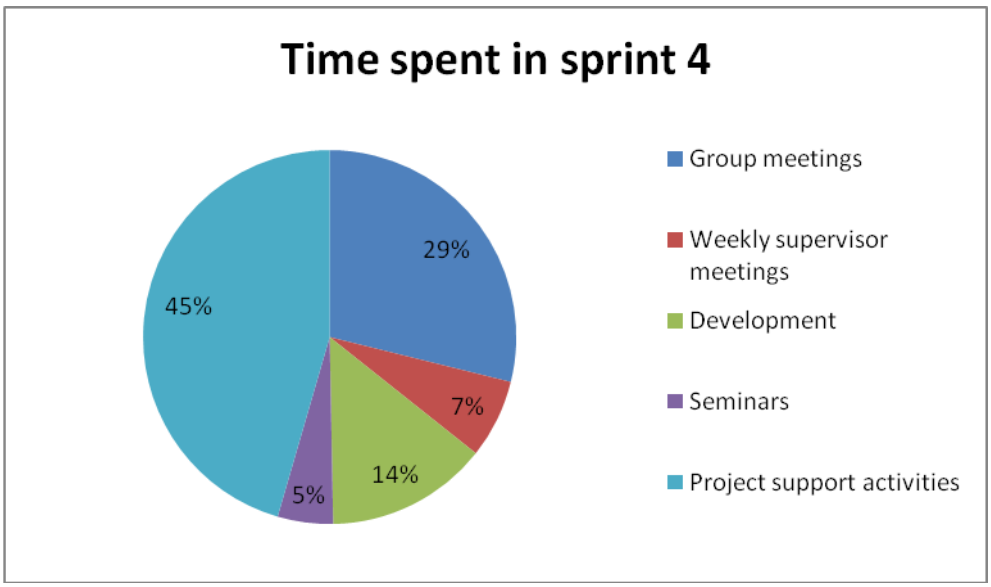


Figure B.5: Sprint 4 time distribution

Stories ⊕ **Tasks**	**Estimated hours**	**Actual hours**
Group meetings	56	75
Weekly supervisor meetings	12	17
Development	60	35
⊕ Adding charting functions		
⊕ Bug fixing		
⊕ Showing simulation time		
⊕ Improving GUI		
Seminars	12	12
Project support activities	100	113.5
⊕ Project report writing		
⊕ Sprint backlog		
⊕ Preparing presentation		
⊕ Class diagrams		
⊕ Weekly report		
⊕ Write meeting minutes		
Summary	**240**	**252.5**

B.1.6. After work

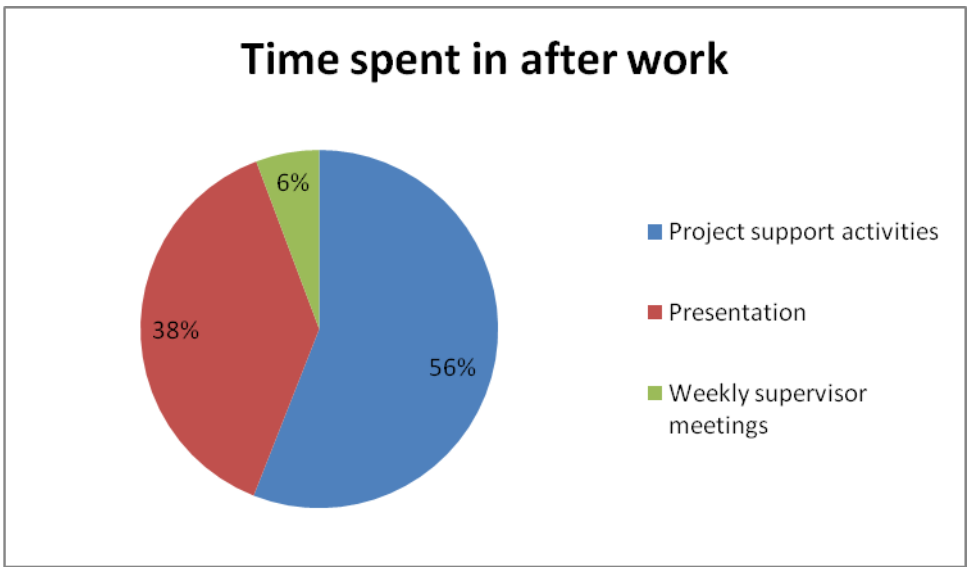


Figure B.6: After work time distribution

Stories ⊕ Tasks	Estimated hours	Actual hours
Weekly supervisor meetings	12	12
Project support activities	38	117
⊕ Project report writing		
⊕ Sprint backlog		
⊕ Class diagrams		
⊕ Proof reading report		
⊕ Write meeting minutes		
Presentation	30	80
⊕ Preparing presentation slides		
⊕ Preparing application demo		
⊕ Presentation rehearsal		
Summary	**80**	**209**

This major deviation from the estimated hours is the result of spending the last weekend before delivery on final preparation of report and presentation.

The weekends were not included in the initial estimation.

Appendix C

Extended Documentation

C.1. XPlanner

C.1.1. Main parts of XPlanner project (terminology)

A project in XPlanner is divided into many parts. These parts help the project manager make better estimations and planning of the project.

This section describes the most important parts of XPlanner's project structure, as well as the interconnection between these parts and user roles in a project.

XPlanner's project structure:

- Project – This is the main work item. XPlanner supports multiple projects for easier handling of large scale projects consisting of several smaller ones. This is the top level entry. For our project we have only one project entry.
- Iteration – This is the skeleton of a project containing many user stories relevant for this iteration. Duration should be between 1 to 3 weeks. In Scrum terminology, an iteration is equivalent to a sprint.
- User story – It's defining comprehensive functionality of program. It's usually written by a customer or defined from requirement specification. Stories are split into tasks.
- Tasks – This is a part of the story which could be done by one or two programmers. It provides part of functionality for fulfilling needs of the whole story. Ideally a case lasts 1 to 3 programming days. It's possible to assign a programmer or a team of programmers for each task.

User roles in XPlanner:

- Customer – The person who defines the user stories.

- Tracker (for each task) – Someone who will probably work on a task or help the developers who work on this task. Usually it's one of the developers.
- Acceptor (for each task) – The developer who apply for this task and is responsible to solve it.

C.1.2. Structure of project and user interface

This section describes the structure of an XPlanner project and how to navigate through the stories and tasks and clarify the understanding of it. It will show some screens of the user interface with comments and description of the main functionality.

C.1.2.1.a Structure

XPlanner has a tree based structure based on the parts which are described in previous chapter (C.1.1). You can navigate through this structure by clicking on the links. Figure C.1 shows the structure.

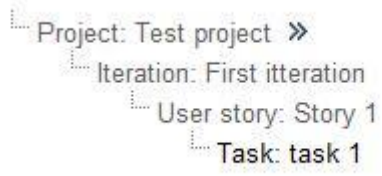


Figure C.1: Project structure

C.1.2.1.b User interface and main functionality

As mentioned in the overview, XPlanner has a web-based interface which allows you to control everything through a web browser. Everything starts on login page, which require your credentials (name and password). After the login process you will see the main page with the projects in which you can participate. See Figure C.2.

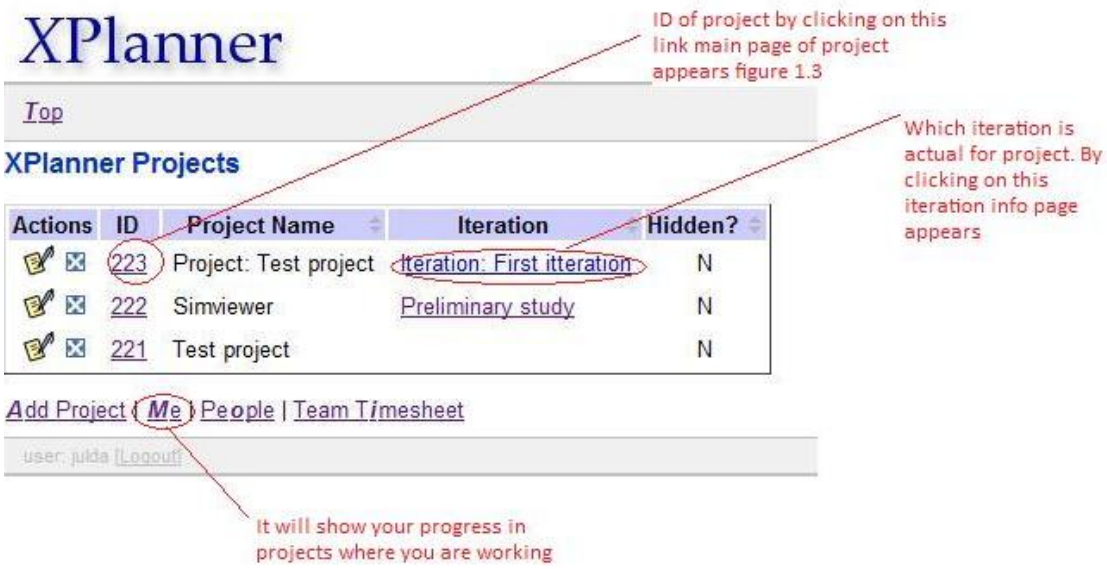


Figure C.2: XPlanner main page

When you enter the main page you can immediately access actual iterations of the project, project main page or information about your progress in the projects. Next figure shows the main page of a project, which includes overview information about the project and a list of all the iterations. It displays basic information such as starting time and ending time of an iteration, days of work for each iteration and the number of user stories. See Figure C.3.

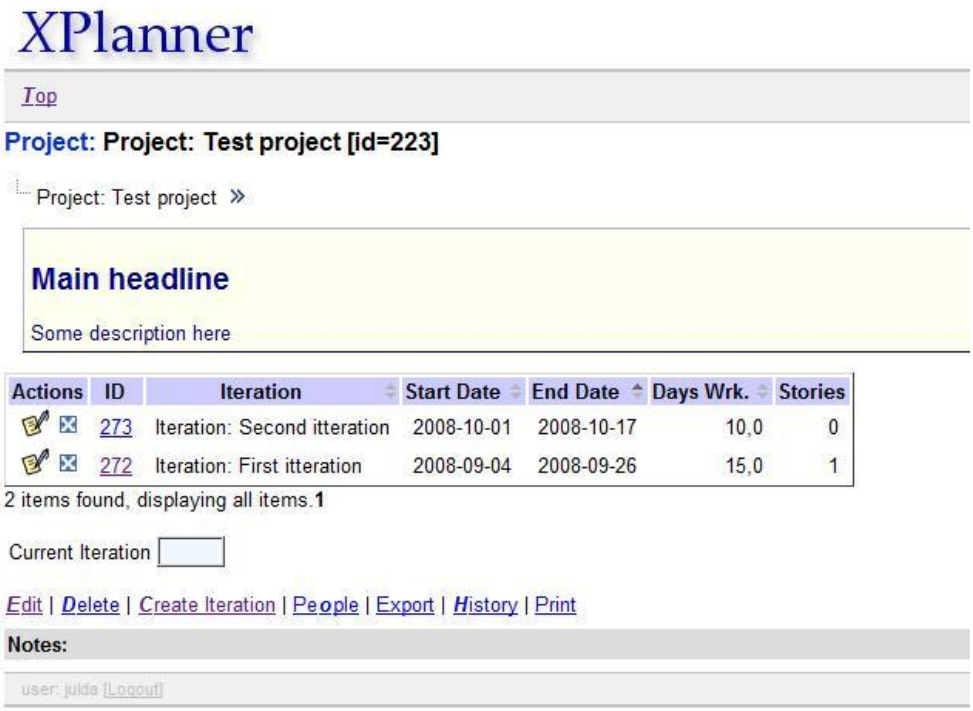

XPlanner
Top
Project: Project: Test project [id=223]
Project: Test project »
Main headline
Some description here
Actions ID Iteration Start Date End Date Days Wrk. Stories
273 Iteration: Second itteration 2008-10-01 2008-10-17 10,0 0
272 Iteration: First itteration 2008-09-04 2008-09-26 15,0 1
2 items found, displaying all items.1
Current Iteration
Edit | Delete | Create Iteration | People | Export | History | Print
Notes:
user: julda [Logout]

Figure C.3: Main page for the project

The pages for iteration information and user story information look similar. Each page can contain some information about an iteration or a story, list of stories if you are on an iteration page and list of tasks if you are on a user story page. It follows the structure of the tree described in C.1.2. They also contain the progress of each story/task (estimated hours, actually completed hours, hours to go etc.). See Figure C.4 for an example of a story information page.

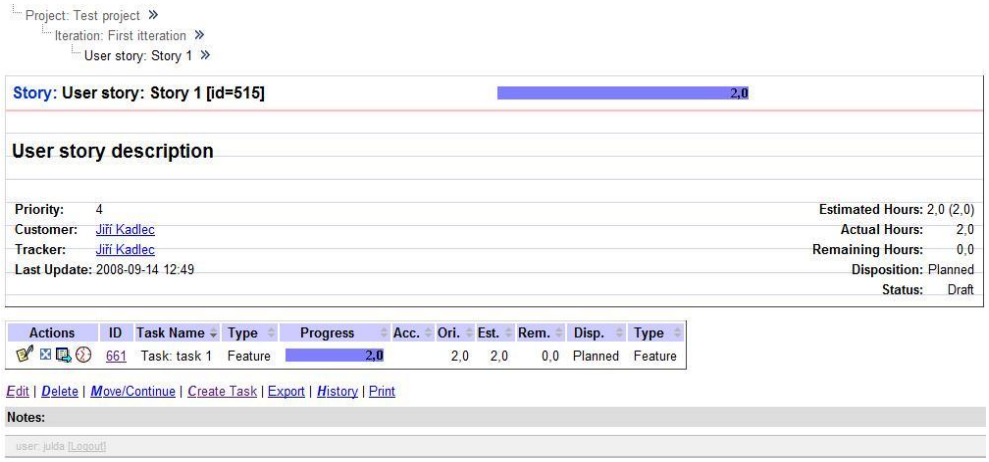

Project: Test project »
Iteration: First itteration »
User story: Story 1 »
Story: User story: Story 1 [id=515]
2,0
User story description
Priority: 4
Customer: Jiří Kadlec
Tracker: Jiří Kadlec
Last Update: 2008-09-14 12:49
Estimated Hours: 2,0 (2,0)
Actual Hours: 2,0
Remaining Hours: 0,0
Disposition: Planned
Status: Draft
Actions ID Task Name Type Progress Acc. Ori. Est. Rem. Disp. Type
661 Task: task 1 Feature 2,0 2,0 2,0 0,0 Planned Feature
Edit | Delete | Move/Continue | Create Task | Export | History | Print
Notes:

Figure C.4: Story information page

From the story information page you can access the task page, where you can assign people to the task. You can also write down the time which they spent on each task. XPlanner keeps track of hours for each person, so it's possible to get an overview of how many hours each person spent on the project. This information provides a really good basis for the project manager to make new estimations, motivational talks and decisions of who should do what etc.

C.1.3. Export functionality

XPlanner also allows you to make exports to different formats.

It supports these formats:

- XML
- MPX
- PDF
- HTML printed out to PDF (not real support)

It looks like the support for export is perfect, but you can't customize the PDF and MPX export without modifying the source, which makes it unsuitable for us. Therefore there is only one way to get fully customizable export; XML + XSLT.

Two of the team members searched for XSLT template for this export, and investigated the need to make our own. The last possibility is to use HTML pages and print directly, but this is not an optimal solution concerning efficiency and readability.

C.1.4. Technical information

XPlanner is an open-source project built on Java and runs like a server application on the Apache Tomcat web server. It has the following requirements:

- Java runtime at least JDK 1.4
- Tomcat application server
- Database engine HSQLDB is the default, but it's also possible to use another database i.e. MySQL

The main technical advantage is that it's 100% Java and platform independent. Our XPlanner runs on Open Suse linux 10.2.